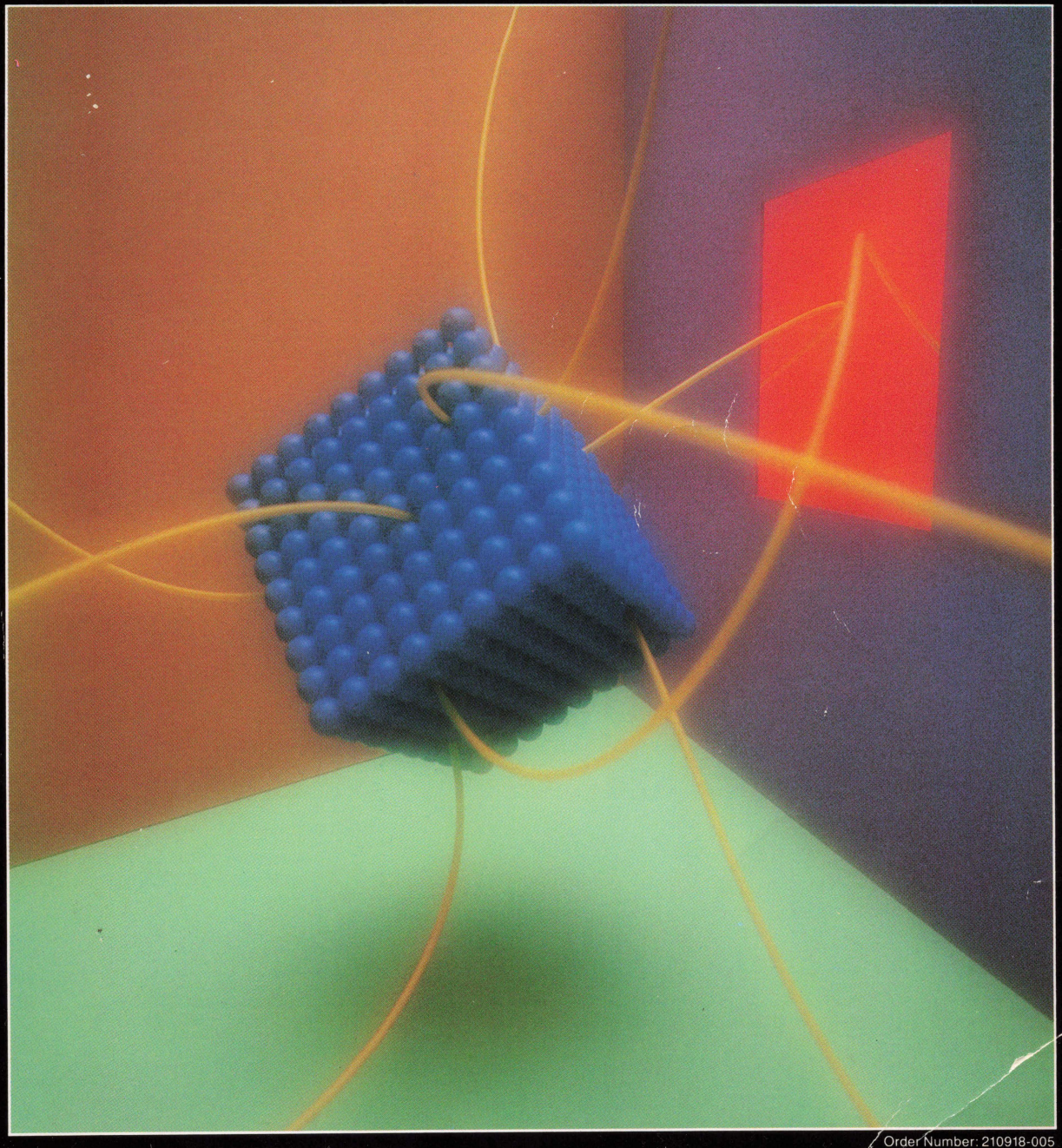
intel®

Embedded Controller Handbook



Order Number: 210918-005

LITERATURE

To order Intel literature write or call:

Intel Literature Sales
P.O. Box 58130
Santa Clara, CA 95052-8130

Intel Literature:
(800) 548-4725

Use the order blank on the facing page or call our **Toll Free** Number listed above to order literature. Remember to add your local sales tax and a 10% postage charge for U.S. and Canada customers, 20% for outside U.S. customers.

1987 HANDBOOKS

Product line handbooks contain data sheets, application notes, article reprints and other design information.

NAME	ORDER NUMBER	*PRICE IN U.S. DOLLARS
COMPLETE SET OF 9 HANDBOOKS Save $50.00 off the retail price of $175.00	231003	**$125.00**
MEMORY COMPONENTS HANDBOOK	210830	**$18.00**
MICROCOMMUNICATIONS HANDBOOK	231658	**$20.00**
EMBEDDED CONTROLLER HANDBOOK (includes Microcontrollers and 8085, 80186, 80188)	210918	**$18.00**
MICROPROCESSOR AND PERIPHERAL HANDBOOK (2 Volume Set)	230843	**$25.00**
DEVELOPMENT TOOLS HANDBOOK	210940	**$18.00**
DOS DEVELOPMENT SOFTWARE CATALOG	280199	**N/C**
OEM BOARDS AND SYSTEMS HANDBOOK	280407	**$18.00**
MILITARY HANDBOOK	210461	**$18.00**
COMPONENTS QUALITY/RELIABILITY HANDBOOK	210997	**$20.00**
SYSTEMS QUALITY/RELIABILITY HANDBOOK	231762	**$20.00**
PRODUCT GUIDE Overview of Intel's complete product lines	210846	**N/C**
LITERATURE PRICE LIST List of Intel Literature	210620	**N/C**
INTEL PACKAGING OUTLINES AND DIMENSIONS Packaging types, number of leads, etc.	231369	**N/C**

*These prices are for the U.S. and Canada only. In Europe and other international locations, please contact your local Intel Sales Office or Distributor for literature prices.

intel

LITERATURE SALES ORDER FORM

NAME: _____

COMPANY: _____

ADDRESS: _____

CITY: _____ STATE: _____ ZIP: _____

COUNTRY: _____

PHONE NO.: ( ) _____

ORDER NO.	TITLE	QTY.	PRICE	TOTAL

Subtotal _____

Must Add Your Local Sales Tax _____

Must add appropriate postage to subtotal (10% U.S. and Canada, 20% all other) → Postage _____

Total _____

Pay by Visa, MasterCard, American Express, Check, Money Order, or company purchase order payable to Intel Literature Sales. Allow 2-4 weeks for delivery.

☐ Visa ☐ MasterCard ☐ American Express Expiration Date _____

Account No. _____

Signature: _____

Mail To: Intel Literature Sales
P.O. Box 58130
Santa Clara, CA
95052-8130

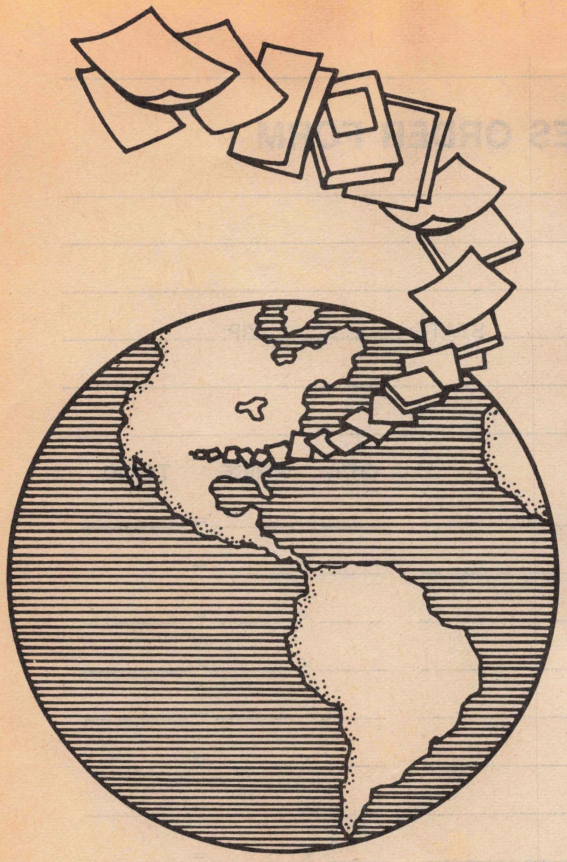
International Customers outside the U.S. and Canada should contact their local Intel Sales Office or Distributor listed in the back of most Intel literature.

European Literature Order Form in back of book.

Call Toll Free: (800) 548-4725 for phone orders

Prices good until 12/31/87.

Source HB



We Bring Our World to Your Door

Intel's New Product Literature Subscription Service.

Keeping up with today's technology takes a lot of time and effort. With Intel's new Literature Subscription Service you will receive a package of current literature plus automatic quarterly updates on all the latest product and service news from Intel. From microprocessors — to peripherals and memories — to OEM boards, systems and software, you can choose to receive information from one, or all three, product categories for an entire year at a low one-time cost.

Save Time and Money. Subscribe Today.

Save 10% when ordering two or more packages.

To order, use the literature order form provided in this book or call TOLL FREE 800-548-4725

The charge for this service covers our printing, postage and handling costs only.

intel®

Each Literature Package Contains:

Newly published Data Sheets, Fact Sheets, Application Notes, Reliability Reports, Errata Reports, Article Reprints, Promotional Offers, Brochures, Flyers, Benchmark Reports, Technical Papers and more...

In Addition, Each Individual Package Contains:

1

Microprocessors, etc.

Product Line Handbooks on Microprocessors, Development Tools and Embedded Controllers.
plus
Quality/Reliability Information, The Product Guide, Literature Guide and Packaging Information.
plus
Three Quarterly Updates containing all new documentation on these products.
(Retail Value of Handbooks alone: $81)

Your price for the complete package with quarterly updates: $70

Order Number 555100

2

Peripherals, Memories, etc.

Product Line Handbooks on Peripherals, Microcommunications, Memories and EPLD
plus
Quality/Reliability Information, The Product Guide, The Literature Guide, Packaging Information and other supporting information.
plus
Three Quarterly Updates containing all new documentation on these products.
(Retail value of Handbooks alone: $83)

Your price for the complete package with quarterly updates: $70

Order Number 555101

3

OEM Boards, Systems and Software

Product Line Handbooks on OEM Boards and Systems
plus
Quality/Reliability Information, The Product Guide, The Literature Guide, and other supporting information.
plus
Three Quarterly Updates containing all new documentation on these products.
(Retail Value of Handbooks alone: $38)

Your price for the complete package with quarterly updates: $60

Order Number 555102

Customers outside the U.S. and Canada should order directly from the U.S. on the U.S. literature order form.

Offer expires 12/31/87

EMBEDDED CONTROLLER HANDBOOK

1987

About Our Cover:

Industry-standard architecture, lots of existing software, design flexibility and upgradeability are just a few features making Intel's microcontroller family the most popular in the industry. Here you see an abstraction of a microcontroller as the center or nucleus of a system connected to many different input/output lines.

Concept/Design: Hall Kelley, Concept/Photography: R. J. Muna

intel

Intel Corporation makes no warranty for the use of its products and assumes no responsibility for any errors which may appear in this document nor does it make a commitment to update the information contained herein.

Intel retains the right to make changes to these specifications at any time, without notice.

Contact your local sales office to obtain the latest specifications before placing your order.

The following are trademarks of Intel Corporation and may only be used to identify Intel Products:

Above, BITBUS, COMMputer, CREDIT, Data Pipeline, FASTPATH, GENIUS, i, î, ICE, iCEL, iCS, iDBP, iDIS, I^2ICE, iLBX, Inboard, i_m, iMDDX, iMMX, Insite, Intel, int$_e$l, int$_e$lBOS, Intelevision, int$_e$ligent Identifier, int$_e$ligent Programming, Intellec, Intellink, iOSP, iPDS, iPSC, iRMX, iSBC, iSBX, iSDM, iSXM, KEPROM, Library Manager, MAP-NET, MCS, Megachassis, MICROMAINFRAME, MULTIBUS, MULTICHANNEL, MULTIMODULE, MultiSERVER, ONCE, OpenNET, OTP, PC-BUBBLE, Plug-A-Bubble, PROMPT, Promware, QUEST, QueX, Quick-Pulse Programming, Ripplemode, RMX/80, RUPI, Seamless, SLD, UPI, and VLSiCEL, and the combination of ICE, iCS, iRMX, iSBC, iSBX, iSXM, MCS, or UPI and a numerical suffix 4-SITE.

Ethernet is a trademark of Xerox.

MDS is an ordering code only and is not used as a product name or trademark. MDS® is a registered trademark of Mohawk Data Sciences Corporation.

*MULTIBUS is a patented Intel bus.

Additional copies of this manual or other Intel literature may be obtained from:

Intel Literature Department
SC6-59
P.O. Box 58065
Santa Clara, CA 95052-8065

© INTEL CORPORATION 1987

Table of Contents

Alphanumeric Index viii
Preface xiii

MCS®-48 FAMILY

Chapter 1
MCS®-48 Single Component System 1-1

Chapter 2
MCS®-48 Expanded System 2-1

Chapter 3
MCS®-48 Instruction Set 3-1

Chapter 4
DATA SHEETS
8243 MCS®-48 Input/Output Expander 4-1
8048AH/8035AHL/8049AH/8039AHL/8050AH/8040AHL HMOS Single-Component 8-Bit Microcomputer 4-8
8748H/8035H/8749H/8039H HMOS-E Single-Component 8-Bit Microcomputer .. 4-18
MCS®-48 Express 4-30

MCS®-51 FAMILY

Chapter 5
Architectural Overview of the MCS®-51 Family of Microcontollers 5-1

Chapter 6
Hardware Description of the 8051 and 8052 6-1

Chapter 7
MCS®-51 Programmer's Guide and Instruction Set 7-1

Chapter 8
DATA SHEETS
8051/8051AH/8052AH/8751H/8031/8031AH/8032AH 8-Bit HMOS and HMOS EPROM Microcontollers 8-1
8751BH 8-Bit HMOS EPROM Microcontroller 8-15
8032BH/8052BH/8752BH 8-Bit HMOS and HMOS EPROM Microcontollers 8-26
80C31BH/80C51BH 8-Bit CHMOS Microcontrollers 8-37
87C51 8-Bit CHMOS EPROM Microcontroller 8-49
87C252 8-Bit CHMOS EPROM Microcontroller 8-62
8051AH/8052AH/8751H/8031AH/8032AH Express 8-66
80C51BH/80C31BH Express 8-68

Chapter 9
APPLICATION NOTES
Using the Intel MCS®-51 Boolean Processing Capabilities AP-70 9-1
Designing with the 80C51BH AP-252 9-46

Chapter 10
ARTICLE REPRINTS
Built-In Basic Interpreter Turns Controller into Versatile System Core AR-374 10-1
Increased Functions in Chip Result in Lighter, Less Costly Portable Computer AR-409 10-9
Using the 8051 Microcontroller with Resonant Transducers 10-14

MCS®-80/85 FAMILY

Chapter 11
DATA SHEETS
8080A/8080A-1/8080A-2 8-Bit N-Channel Microprocessor 11-1
8085AH/8085AH-2/8085AH-1 8-Bit HMOS Microprocessors 11-11
8155H/8156H/8155H-2/8156H-2 2048-Bit Static HMOS RAM with I/O Ports and Timer 11-31

Table of Contents (Continued)

8185/8185-2 1024 x 8-Bit Static RAM for MCS®-85 11-45
8224 Clock Generator and Driver for 8080A CPU 11-50
8228/8238 System Controller and Bus Driver for 8080A CPU 11-55
8257/8257-5 Programmable DMA Controller 11-59
8755A 16,384-Bit EPROM with I/O 11-80

MCS®-96 FAMILY

Chapter 12
MCS®-96 Architectural Overview 12-1

Chapter 13
MCS®-96 Instruction Set 13-1

Chapter 14
MCS®-96 Hardware Design Information 14-1

Chapter 15
DATA SHEETS
809X-90, 839X-90 15-1
809X-90, 839X-90 Express 15-19
809XBH, 839XBH, 879XBH with 8- or 16-Bit External Bus 15-24

Chapter 16
APPLICATION NOTES
Using the 8096 AP-248 16-1
An FFT Algorithm for MCS®-96 Products Including Supporting Routines and Examples AP-275 16-103

80186 FAMILY

Chapter 17
DATA SHEETS
80186 High Integration 16-Bit Microprocessor 17-1
80188 High Integration 8-Bit Microprocessor 17-53
82188, Integrated Bus Controller for 8086, 8088, 80186, 80188 Processors 17-107

Chapter 18
APPLICATION NOTES
Introduction to the 80186 Microprocessor AP-186 18-1
High Speed Numerics with the 80186/80188 and 8087 AP-258 18-78
80186/188 Interface to Intel Microcontrollers AP-286 18-94

THE RUPI™ FAMILY

Chapter 19
The RUPI™-44 Family 19-1

Chapter 20
8044 Architecture 20-1

Chapter 21
8044 Serial Interface 21-1

Chapter 22
8044 Application Examples 22-1

Chapter 23
DATA SHEET
8044AH/8344AH/8744H High Performance 8-Bit Microcontroller with On-Chip Serial Communication Controller 23-1

Chapter 24
ARTICLE REPRINT
Microcontroller with Integrated High Performance Communications Interface 24-1

Table of Contents (Continued)

DESIGN CONSIDERATIONS

Chapter 25

APPLICATION NOTES

Designing Microcontroller Systems for Electrically Noisy Environments AP-125 25-1

Oscillators for Microcontrollers AP-155 25-23

Index 25-55

Alphanumeric Index

80C31BH/80C51BH 8-Bit CHMOS Microcontrollers 8-37
80C51BH/80C31BH Express 8-68
80186 High Integration 16-Bit Microprocessor 17-1
80186/188 Interface to Intel Microcontrollers AP-286 18-94
80188 High Integration 8-Bit Microprocessor 17-53
8032BH/8052BH/8752BH 8-Bit HMOS and HMOS EPROM Microcontollers 8-26
8044 Application Examples 22-1
8044 Architecture 20-1
8044 Serial Interface 21-1
8044AH/8344AH/8744H High Performance 8-Bit Microcontroller with On-Chip Serial Communication Controller 23-1
8048AH/8035AHL/8049AH/8039AHL/8050AH/8040AHL HMOS Single-Component 8-Bit Microcomputer 4-8
8051/8051AH/8052AH/8751H/8031/8031AH/8032AH 8-Bit HMOS and HMOS EPROM Microcontollers 8-1
8051AH/8052AH/8751H/8031AH/8032AH Express 8-66
8080A/8080A-1/8080A-2 8-Bit N-Channel Microprocessor 11-1
8085AH/8085AH-2/8085AH-1 8-Bit HMOS Microprocessors 11-11
809X-90, 839X-90 15-1
809X-90, 839X-90 Express 15-19
809XBH, 839XBH, 879XBH with 8- or 16-Bit External Bus 15-24
8155H/8156H/8155H-2/8156H-2 2048-Bit Static HMOS RAM with I/O Ports and Timer 11-31
8185/8185-2 1024 x 8-Bit Static RAM for MCS®-85 11-45
82188, Integrated Bus Controller for 8086, 8088, 80186, 80188 Processors 17-107
8224 Clock Generator and Driver for 8080A CPU 11-50
8228/8238 System Controller and Bus Driver for 8080A CPU 11-55
8243 MCS®-48 Input/Output Expander 4-1
8257/8257-5 Programmable DMA Controller 11-59
87C252 8-Bit CHMOS EPROM Microcontroller 8-62
87C51 8-Bit CHMOS EPROM Microcontroller 8-49
8748H/8035H/8749H/8039H HMOS-E Single-Component 8-Bit Microcomputer 4-18
8751BH 8-Bit HMOS EPROM Microcontroller 8-15
8755A 16,384-Bit EPROM with I/O 11-80
An FFT Algorithm for MCS®-96 Products Including Supporting Routines and Examples AP-275 16-103
Built-In Basic Interpreter Turns Controller into Versatile System Core AR-374 10-1
Designing with the 80C51BH AP-252 9-46
Designing Microcontroller Systems for Electrically Noisy Environments AP-125 25-1
Hardware Description of the 8051 and 8052 6-1
High Speed Numerics with the 80186/80188 and 8087 AP-258 18-78
Increased Functions in Chip Result in Lighter, Less Costly Portable Computer AR-409 10-9
Introduction to the 80186 Microprocessor AP-186 18-1
Microcontroller with Integrated High Performance Communications Interface 24-1
MCS®-48 Express 4-30
MCS®-96 Architectural Overview 12-1
MCS®-96 Hardware Design Information 14-1
MCS®-96 Instruction Set 13-1
Oscillators for Microcontrollers AP-155 25-23
The RUPI™-44 Family 19-1
Using the Intel MCS®-51 Boolean Processing Capabilities AP-70 9-1
Using the 8051 Microcontroller with Resonant Transducers 10-14
Using the 8096 AP-248 16-1

CUSTOMER SUPPORT

CUSTOMER SUPPORT

Customer Support is Intel's complete support service that provides Intel customers with hardware support, software support, customer training, and consulting services. For more information contact your local sales offices.

After a customer purchases any system hardware or software product, service and support become major factors in determining whether that product will continue to meet a customer's expectations. Such support requires an international support organization and a breadth of programs to meet a variety of customer needs. As you might expect, Intel's customer support is quite extensive. It includes factory repair services and worldwide field service offices providing hardware repair services, software support services, customer training classes, and consulting services.

HARDWARE SUPPORT SERVICES

Intel is committed to providing an international service support package through a wide variety of service offerings available from Intel Hardware Support.

SOFTWARE SUPPORT SERVICES

Intel's software support consists of two levels of contracts. Standard support includes TIPS (Technical Information Phone Service), updates and subscription service (product-specific troubleshooting guides and COMMENTS Magazine). Basic support includes updates and the subscription service. Contracts are sold in environments which represent product groupings (i.e., iRMX environment).

CONSULTING SERVICES

Intel provides field systems engineering services for any phase of your development or support effort. You can use our systems engineers in a variety of ways ranging from assistance in using a new product, developing an application, personalizing training, and customizing or tailoring an Intel product to providing technical and management consulting. Systems Engineers are well versed in technical areas such as microcommunications, real-time applications, embedded microcontrollers, and network services. You know your application needs; we know our products. Working together we can help you get a successful product to market in the least possible time.

CUSTOMER TRAINING

Intel offers a wide range of instructional programs covering various aspects of system design and implementation. In just three to ten days a limited number of individuals learn more in a single workshop than in weeks of self-study. For optimum convenience, workshops are scheduled regularly at Training Centers worldwide or we can take our workshops to you for on-site instruction. Covering a wide variety of topics, Intel's major course categories include: architecture and assembly language, programming and operating systems, bitbus and LAN applications.

CUSTOMER SUPPORT

CUSTOMER SUPPORT

Customer Support is Intel's complete support service that provides Intel customers with hardware support, software support, customer training, and consulting services. For more information contact your local sales offices.

After a customer purchases any system hardware or software product, service and support become major factors in determining whether that product will continue to meet a customer's expectations. Such support requires an international support organization and a breadth of programs to meet a variety of customer needs. As you might expect, Intel's customer support is quite extensive. It includes factory repair services and worldwide field service offices providing hardware repair services, software support services, customer training classes, and consulting services.

HARDWARE SUPPORT SERVICES

Intel is committed to providing an international service support package through a wide variety of service offerings available from Intel Hardware Support.

SOFTWARE SUPPORT SERVICES

Intel's software support consists of two levels of contracts: Standard support includes TIPS (Technical Information Phone Service), updates and subscription service (product-specific trouble-shooting guides and CONFIDENCE Magazine). Basic support includes updates and the subscription service. Contracts are sold in one-year periods. Software support is available for most Intel development software products and for other Intel software products (i.e., iRMX environment).

CONSULTING SERVICES

Intel provides field systems engineering services for any phase of your development or support effort. You can use our systems engineers in a variety of ways ranging from assistance in using a new product, developing an application, personalizing training, and customizing or tailoring an Intel product to providing technical and management consulting. Systems Engineers are well versed in technical areas such as microcommunications, real-time applications, embedded microcontrollers, and network services. You know your application needs, we know our products. Working together we can help you get a successful product to market in the least possible time.

CUSTOMER TRAINING

Intel offers a wide range of instructional programs covering various aspects of system design and implementation. In just three to ten days a limited number of individuals learn more in a single workshop than in weeks of self-study. For optimum convenience, workshops are scheduled regularly at Training Centers worldwide or we can take our workshops to you for on-site instruction. Our study a wide variety of topics, Intel's major course categories include architecture and assembly language, programming and operating systems, buses and LAN applications.

Preface

Preface

PREFACE

Computer systems can be characterized as being "reprogrammable" or "embedded". Reprogrammable systems are those that look and behave like computers to the ultimate user. An obvious example is the personal computer. These systems contain some form of mass storage device which stores a number of different computer programs that the user may call up and use as required. The input and output devices attached to these systems are there to communicate with the user. Embedded systems, as the name implies, are contained within a final product which may not look or feel like a computer to the end user. An example here is a computer which controls the ubiquitous office copier machine. These systems rarely contain mass storage; the programs are stored in ROM or EPROM devices. The input and output devices attached are not limited to communication with the user (e.g. the control panel); they also monitor and control mechanisms and processes within the device (e.g. the paper feed mechanism).

Embedded control applications can be broken into three broad categories; sequential control, closed loop control, and data control. Sequential control deals with the control and monitoring of a system as a sequence of events; activate the paper feed roller, wait for the paper feed indicator then activate the drum mechanism. Closed loop control involves closely monitoring the output of process or device and altering its inputs to achieve the desired output; if the feed roller motor is turning too slowly or is decelerating then increase the drive to the motor to compensate. These two categories involve fixed programs that interface directly with the outside world. The data structures involved are normally small and simple. The third category of embedded control (data control) still runs fixed programs but the interface to the outside word becomes more indirect and the data structures become larger and more complex. A high end copier might digitize an image and then use image processing techniques to enhance contrast and then scale and rotate the image to fit the paper being used.

During 1986 Intel consolidated four of its product families intended for embedded control applications into the Embedded Control Operation.

MCS®-48: Designed for general purpose 8-bit sequential control applications. Versions of these parts are available with program storage and up to 256 bytes of data storage onboard. The maximum program size is 4 Kbytes.

MCS®-51: Designed for advanced 8-bit sequential control applications. These parts are similar to the MCS®-48 family parts but operate from 2 to 5 times faster and include more on-board peripherals. A unique feature of the MCS®-51 family is a built in Boolean processor which performs calculations on Boolean (one bit) variables. The maximum program size is 64 Kbytes.

MCS®-96: Designed for advanced 16-bit closed loop control applications. These parts include a processor capable of high performance integer arithmetic and 232 bytes of general purpose registers which can be used for byte, word, or double-word operands. A separate subsystem manages timer functions. Versions are available with on-board A/D conversion and 8 Kbytes of on-board program memory. A unique feature of the newer (BH) members of the family is dynamic bus sizing which allows the parts to operate on both 8-bit and 16-bit busses. The maximum program size is 64 Kbytes.

80186/80188: These parts are highly integrated versions of the 8086 microprocessor intended for data control applications. They combine 15 to 20 of the most common 8086 system components onto one device. Included with the CPU is the clock generator, an interrupt controller, timers, DMA channels and chip select logic. The 80186 operates on a 16-bit bus and the 80188 operates on an 8-bit bus. Both parts operate on a 16-bit bus internally.

Part of the motivation for combining all of these products into one handbook was the realization that, although the categorization of embedded control applications into the three segments (sequential, closed loop, and data control) is useful for conceptualizing application requirements, real applications of these parts contain attributes from all three categories. Combining these product lines into one handbook will make it easier for customers involved with embedded applications to find the information they require.

The remainder of this chapter will give a very brief overview of each of the four products discussed. The remainder of this book is intended to provide detailed technical information on Intel's embedded control product line.

MCS®-48 MICROCONTROLLERS

Intel's MCS-48 family of 8-bit microcontrollers has become a world standard and has been in production for 10 years. They are available in several versions: with on board ROM, on board EPROM, or CPU only, to better fit specific application needs. MCS-48 products are now fabricated on advanced HMOS processes offering higher performance and reliability with less power consumption.

Features common to all members of this family are:

- 8-bit CPU with 1.36 microsecond instruction cycle
- 27 I/O lines
- 8-bit bit timer/counter
- 2 interrupts
- 4 Kbyte maximum program size
- 256 byte maximum on-board RAM size
- 256 byte maximum off-board RAM size

Table 1. MCS®-48 Microcontrollers

Device Name	ROMless Version	EPROM Version	ROM Bytes	RAM Bytes
8048AH	8035AHL	8748H	1K	64
8049AH	8039AHL	8749H	2K	128
8050AH	8040AHL	—	4K	256

Intel has ten years of experience in manufacturing both the EPROM and ROM versions of this chip. It provides a low cost solution to applications such as keyboards, low end printers, and electronic carburator control.

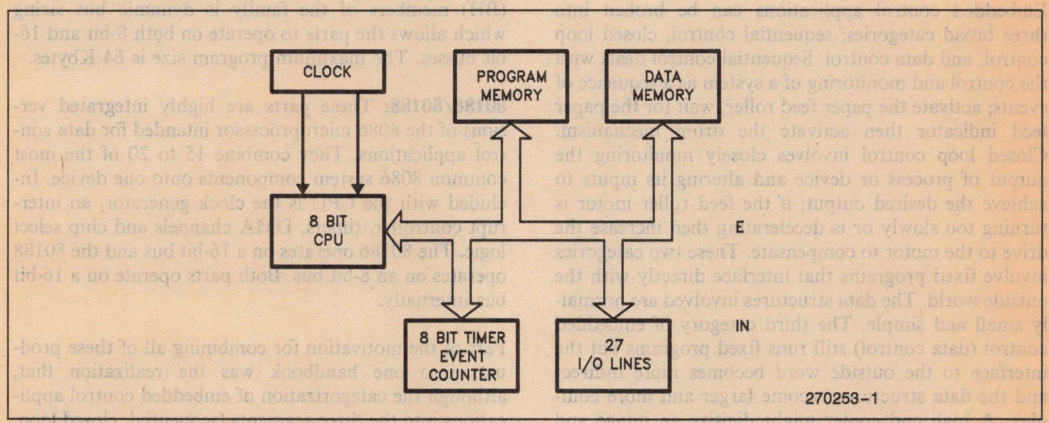


Figure 1. MCS®-48 Block Diagram

MCS®-51 MICROCONTROLLERS

Intel's MCS-51 family is the industry standard for 8-bit high performance microcontrollers. The family architecture is optimized for sequential real-time control applications. They are available in several versions with on-board ROM, on-board EPROM, and CPU only to better fit specific application needs. MCS-51 products are available on advanced HMOS and CHMOS processes.

Features common to all members of this family are:

- 8-bit CPU optimized for control applications
- Extensive Boolean processing (single-bit logic) capabilities
- 32 bidirectional and individually addressable I/O lines
- Full-Duplex UART
- 5 source interrupt structure with 2 priority levels (6 sources on parts with 3 timer/counters)
- 64 Kbyte maximum program size
- 256 byte maximum on-board RAM size
- 64 Kbyte maximum off-board RAM size
- Power down/Idle modes for CHMOS parts

The MCS-51 family has found wide acceptance throughout a wide range of applications, ranging from those slightly more complex than a typical MCS-48 application through medical instrumentation and anti-skid braking modules for automobiles.

Table 2. Advanced 8-Bit Microcontrollers

Device Name	ROMless Version	EPROM Version	ROM Bytes	RAM Bytes	16-Bit Timers	Circuit Type
8051	8031	8751H	4K	128	2	HMOS
8051AH	8031AH	8751H	4K	128	2	HMOS
8052AH	8032AH	8752BH	8K	256	3	HMOS
80C51BH	80C31BH	87C51H	4K	128	2	CHMOS

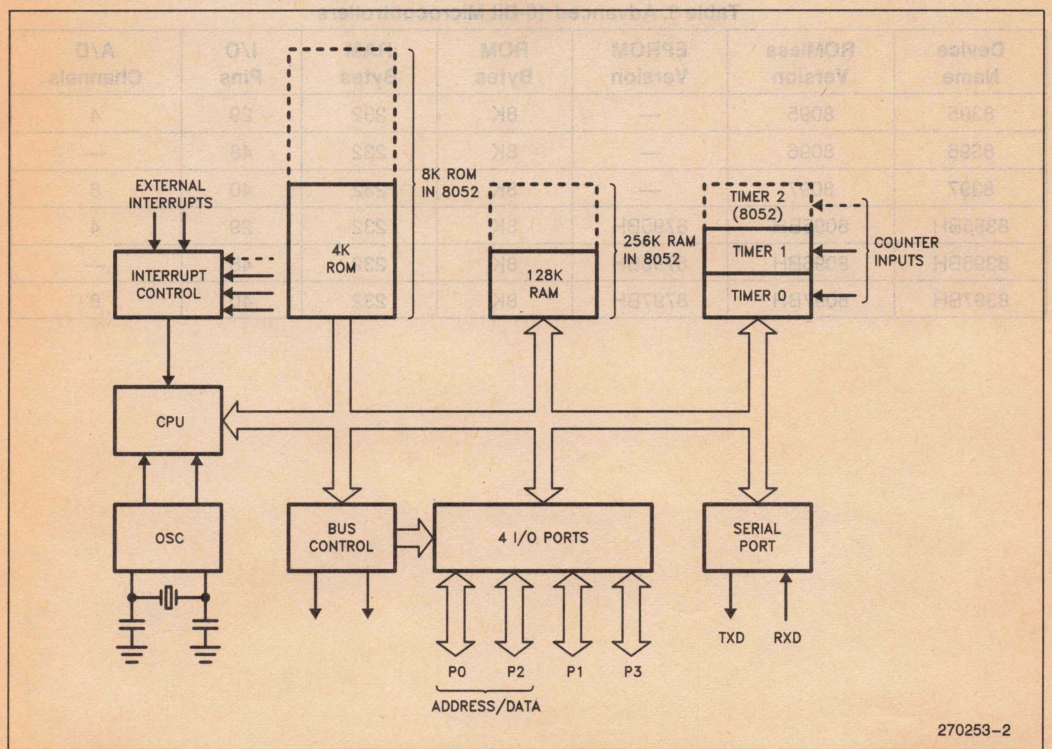


Figure 2. Block Diagram of the 8051/8052AH

MCS®-96 MICROCONTROLLERS

The MCS-96 Family of microcontrollers was designed for applications which combine high performance 16-bit fixed-point arithmetic with an immediate interface to real world devices and events. The architecture is based on a single 64 Kbyte address space. In addition to being accessable as memory, the first 256 bytes of this space are also directly addressable as registers. These locations are on-chip for high performance and can be treated as byte, word, or double-word, operands by the programmer. Twenty-four bytes of these registers are used to control the on-board peripherals; the remaining 232 bytes are usuable by the programmer as general purpose registers. The combination of a register file, on-board I/O, and a high performance 16-bit CPU makes the MCS-96 family well matched to closed loop control applications.

Features common to all members of this family are:

- 16-bit timer (Timer1)
- 16-bit counter (Timer2)
- Full-Duplex UART with independent baud-rate generator
- Watchdog timer
- 8-bit resolution Pulse Width Modulator (PWM)
- 48 I/O lines (33 for 48-pin parts)
- HSIO unit

The HSIO (High Speed Input/Output) unit is an independent timer subsystem which manages Timer1 and Timer2 for the programmer. Events (e.g. setting an I/O pin) can be scheduled to occur automatically when either of the two timers reaches a preset value. External events can be recorded in a FIFO along with the value of Timer1 when the event occurred. The HSIO is connected to eight pins on the 8096 and can generate and monitor events with a 2 microsecond resolution (12 MHz crystal).

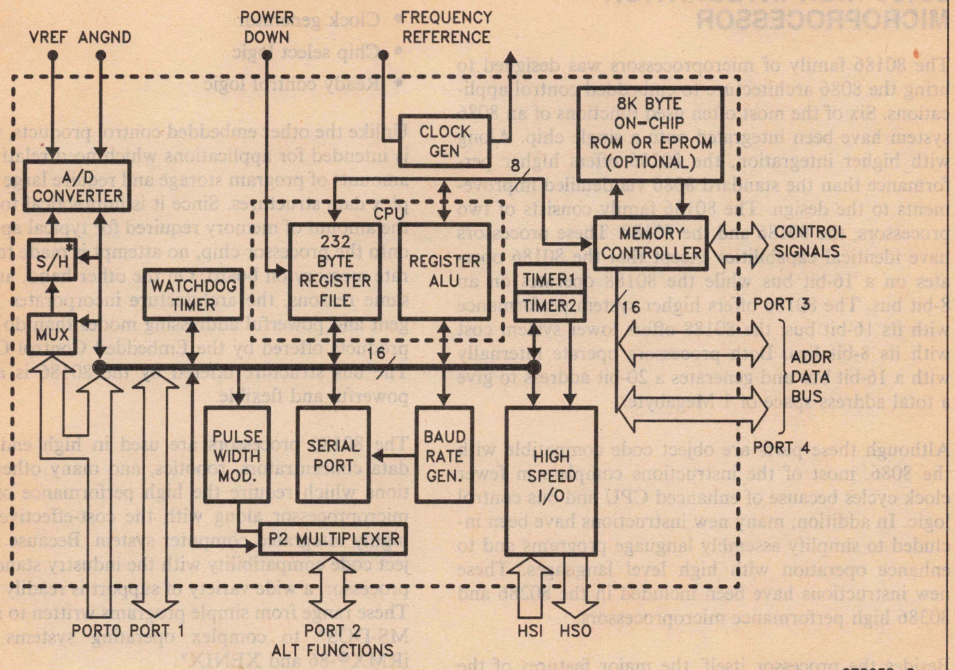
In addition to an EPROM version, the 8x9xBH offer several improvements over the non BH parts:

- Dynamic selection of 8-/16-bit bus operation (versus fixed 16-bit bus)
- Programmable READY logic
- Sample and Hold input to the A/D converter

The MCS-96 family is being used now in a wide variety of complex control tasks such as robotics, motor control, hard disk mechanisms, printers, modems, automotive engine control and high performance anti-skid braking applications.

Table 3. Advanced 16-Bit Microcontrollers

Device Name	ROMless Version	EPROM Version	ROM Bytes	RAM Bytes	I/O Pins	A/D Channels
8395	8095	—	8K	232	29	4
8396	8096	—	8K	232	48	—
8397	8097	—	8K	232	40	8
8395BH	8095BH	8795BH	8K	232	29	4
8396BH	8096BH	8796BH	8K	232	48	—
8397BH	8097BH	8797BH	8K	232	40	8



*Sample and Hold is only present on 8x9xBH devices.

Figure 3. MCS®-96 Block Diagram

80186 HIGH INTEGRATION MICROPROCESSOR

The 80186 family of microprocessors was designed to bring the 8086 architecture to embedded control applications. Six of the most often used functions of an 8086 system have been integrated onto a single chip. Along with higher integration, the 80186 offers higher performance than the standard 8086 via detailed improvements to the design. The 80186 family consists of two processors; the 80186 and the 80188. These processors have identical capabilities except that the 80186 operates on a 16-bit bus while the 80188 operates on an 8-bit bus. The 80186 offers higher system performance with its 16-bit bus; the 80188 offers lower system cost with its 8-bit bus. Both processors operate internally with a 16-bit bus and generates a 20-bit address to give a total address space of 1 Megabyte.

Although these parts are object code compatible with the 8086, most of the instructions complete in fewer clock cycles because of enhanced CPU and bus control logic. In addition, many new instructions have been included to simplify assembly language programs and to enhance operation with high level languages. These new instructions have been included in the 80286 and 80386 high performance microprocessors.

Besides the processor itself, the major features of the 80186 and 80188 include:

- 2 Channel DMA (Direct Memory Access) unit
- 3 16-bit timer/counters
- Interrupt controller
- Clock generator
- Chip select logic
- Ready control logic

Unlike the other embedded control products, the 80186 is intended for applications which need relatively large amounts of program storage and require large and complex data structures. Since it is impractical to integrate the amount of memory required for typical applications onto the processor chip, no attempt is made to incorporate memory on board. On the other hand, and for the same reasons, the architecture incorporates more elegent and powerful addressing modes than do the other products offered by the Embedded Control Operation. The bus structure offered by the 80186 is also more powerful and flexible.

The 80186 processors are used in high end printers, data concentrators, robotics, and many other applications which require the high performance of a 16-bit microprocessor along with the cost-effectiveness of a highly integrated computer system. Because it has object code compatibility with the industry standard 8086 processor, a wide variety of support is readily available. These range from simple programs written to run under MS-DOS* to complex operating systems such as iRMX®-86 and XENIX*.

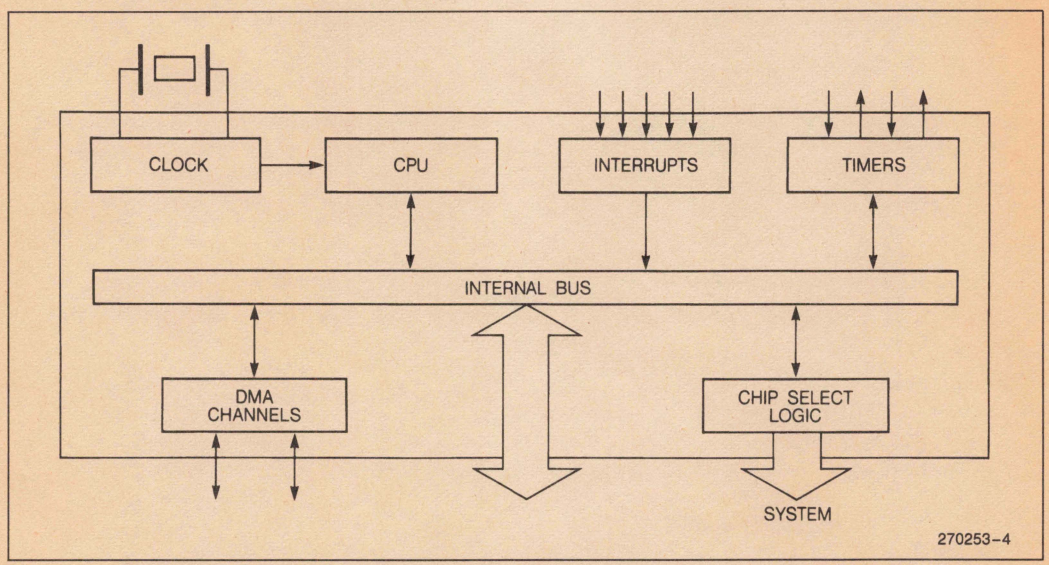


Figure 4. 80186/80188 Block Diagram—A CPU Board On a Single Silicon Chip

*MS-DOS and XENIX are trademarks of Microsoft Corporation.

MCS®-48 Single Component System

1

THE SINGLE COMPONENT MCS®-48 SYSTEM

1.0 INTRODUCTION

Sections 2 through 5 describe in detail the functional characteristics of the 8748H and 8749H EPROM, 8048AH/8049AH/8050AH ROM, and 8035AHL/8039AHL/8040-AHL CPU only single component microcomputers. Unless otherwise noted, details within these sections apply to all versions. This chapter is limited to those functions useful in single-chip implementations of the MCS®-48. The Chapter on the Expanded MCS®-48 System discusses functions which allow expansion of program memory, data memory, and input output capability.

2.0 ARCHITECTURE

The following sections break the MCS-48 Family into functional blocks and describe each in detail. The following description will use the 8048AH as the representative product for the family. See Figure 1.

2.1 Arithmetic Section

The arithmetic section of the processor contains the basic data manipulation functions of the 8048AH and can be divided into the following blocks:

- Arithmetic Logic Unit (ALU)
- Accumulator
- Carry Flag
- Instruction Decoder

In a typical operation data stored in the accumulator is combined in the ALU with data from another source on the internal bus (such as a register or I/o port) and the result is stored in the accumulator or another register.

The following is more detailed description of the function of each block.

INSTRUCTION DECODER

The operation code (op code) portion of each program instruction is stored in the Instruction Decoder and converted to outputs which control the function of each of the blocks of the Arithmetic Section. These lines control the source of data and the destination register as well as the function performed in the ALU.

ARITHMETIC LOGIC UNIT

The ALU accepts 8-bit data words from one or two sources and generates an 8-bit result under control of the Instruction Decoder. The ALU can perform the following functions:

- Add With or Without Carry
- AND, OR, Exclusive OR
- Increment/Decrement
- Bit Complement
- Rotate Left, Right
- Swap Nibbles
- BCD Decimal Adjust

If the operation performed by the ALU results in a value represented by more than 8 bits (overflow of most significant bit), a Carry Flag is set in the Program Status Word.

ACCUMULATOR

The accumulator is the single most important data register in the processor, being one of the sources of input to the ALU and often the destination of the result of operations performed in the ALU. Data to and from I/O ports and memory also normally passes through the accumulator.

2.2 Program Memory

Resident program memory consists of 1024, 2048, or 4096 words eight bits wide which are addressed by the program counter. In the 8748H and the 8749H this memory is user programmable and erasable EPROM; in the 8048AH/8049AH/8050AH the memory is ROM which is mask programmable at the factory. The 8035AHL/8039AHL/8040AHL has no internal program memory and is used with external memory devices. Program code is completely interchangeable among the various versions. To access the upper 2K of program memory in the 8050AH, and other MCS-48 devices, a select memory bank and a JUMP or CALL instruction must be executed to cross the 2K boundary.

There are three locations in Program Memory of special importance as shown in Figure 2.

LOCATION 0
Activating the Reset line of the processor causes the first instruction to be fetched from location 0.

LOCATION 3
Activating the Interrupt input line of the processor (if interrupt is enabled) causes a jump to subroutine at location 3.

LOCATION 7
A timer/counter interrupt resulting from timer counter overflow (if enabled) causes a jump to subroutine at location 7.

Therefore, the first instruction to be executed after initialization is stored in location 0, the first word of an external interrupt service subroutine is stored in location 3, and the first word of a timer/counter service routines

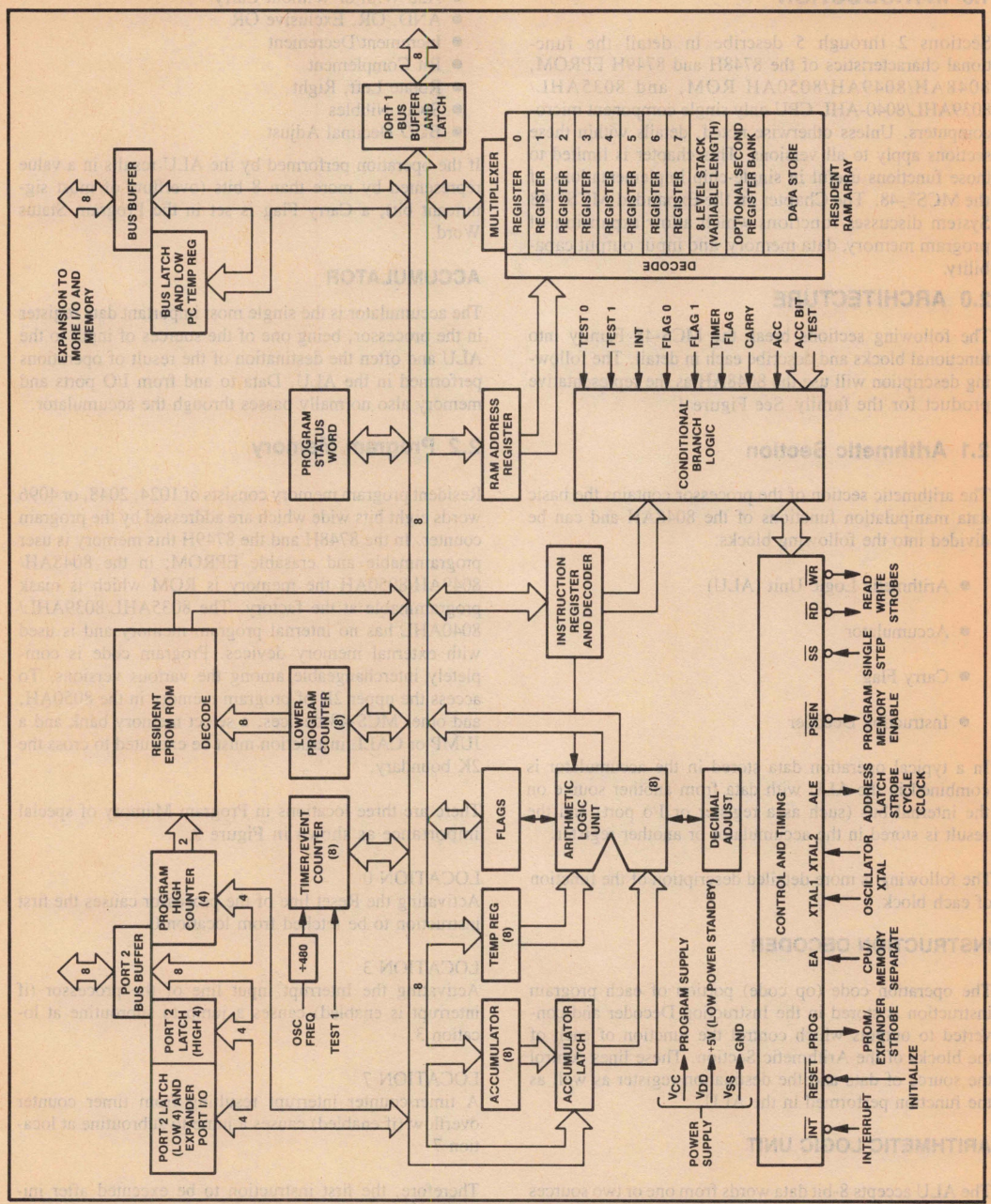


Figure 1. 8748H/8048H/8749AH/8050AH Block Diagram

is stored in location 7. Program memory can be used to store constants as well as program instructions. Instructions such as MOVP and MOVP3 allow easy access to data "lookup" tables.

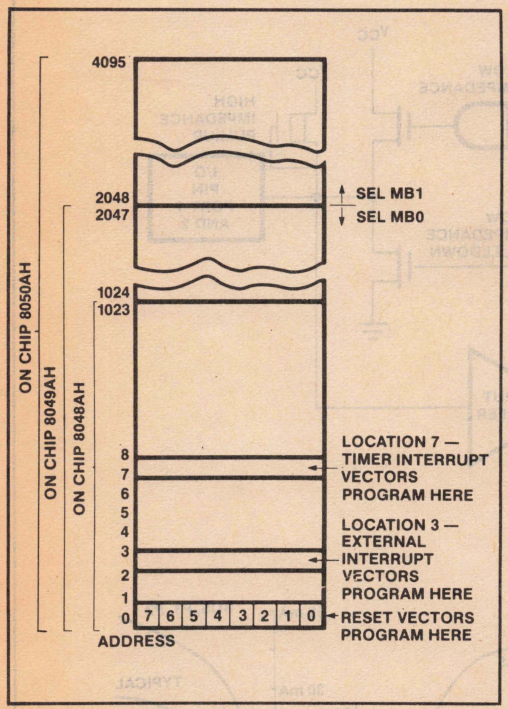


Figure 2. Program Memory Map

2.3 Data Memory

Resident data memory is organized as 64, 128, or 256 by 8-bits wide in the 8048AH, 8049AH and 8050AH. All locations are indirectly addressable through either of two RAM Pointer Registers which reside at address 0 and 1 of the register array. In addition, as shown in Figure 3, the first 8 locations (0–7) of the array are designated as working registers and are directly addressable by several instructions. Since these registers are more easily addressed, they are usually used to store frequently accessed intermediate results. The DJNZ instruction makes very efficient use of the working registers as program loop counters by allowing the programmer to decrement and test the register in a single instruction.

By executing a Register Bank Switch instruction (SEL RB) RAM locations 24–31 are designated as the working registers in place of locations 0–7 and are then directly addressable. This second bank of working registers may be used as an extension of the first bank or reserved for use during interrupt service subroutines allowing the registers of Bank 0 used in the main program to be instantly "saved" by a Bank Switch. Note that if this second bank is not used, locations 24–31 are still addressable as general purpose RAM. Since the two RAM pointer Registers R0 and R1 are a part of the working register array, bank switching effectively creates two more pointer registers (R0/and R1/) which can be used with R0 and R1 to easily access up to four separate working areas in RAM at one time. RAM locations (8–23) also serve a dual role in that they contain the program counter stack as explained in Section 2.6. These locations are addressed by the Stack Pointer during subroutine calls as well as by RAM Pointer Registers R0 and R1. If the level of subroutine nesting is less than 8, all stack registers are not required and can be used as general purpose RAM locations. Each level of subroutine nesting not used provides the user with two additional RAM locations.

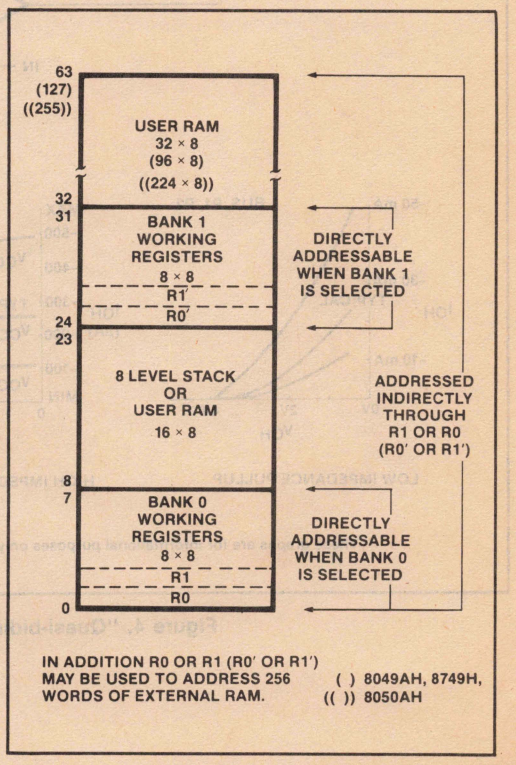
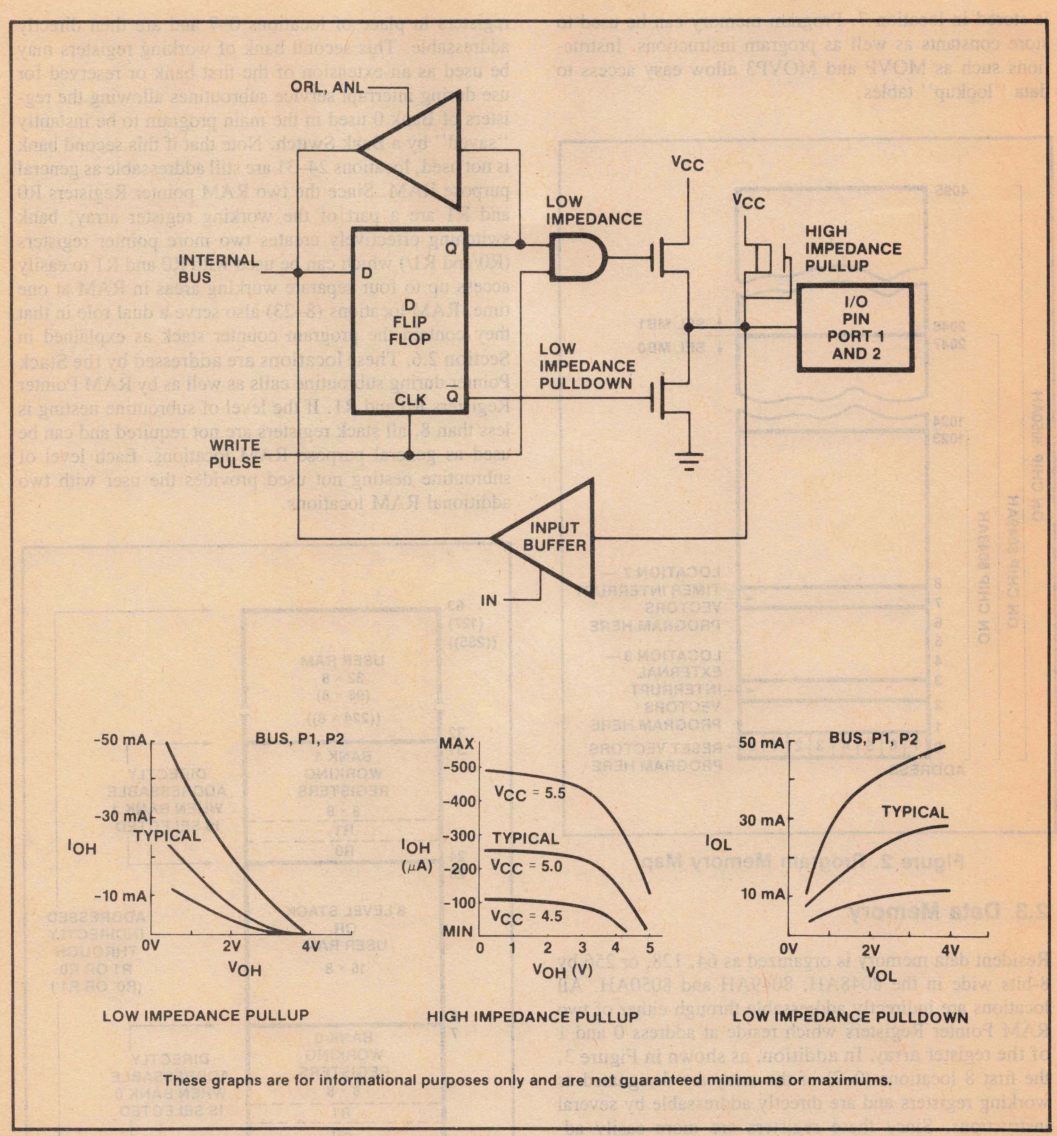


Figure 3. Data Memory Map



These graphs are for informational purposes only and are not guaranteed minimums or maximums.

Figure 4. "Quasi-bidirectional" Port Structure

2.4 Input/Output

The 8048AH has 27 lines which can be used for input or output functions. These lines are grouped as 3 ports of 8 lines each which serve as either inputs, outputs or bidirectional ports and 3 "test" inputs which can alter program sequences when tested by conditional jump instructions.

PORTS 1 AND 2

Ports 1 and 2 are each 8 bits wide and have identical characteristics. Data written to these ports is statically latched and remains unchanged until rewritten. As input ports these lines are non-latching, i.e., inputs must be present until read by an input instruction. Inputs are fully TTL compatible and outputs will drive one standard TTL load.

The lines of ports 1 and 2 are called quasi-bidirectional because of a special output circuit structure which allows each line to serve as an input, and output, or both even though outputs are statically latched. Figure 4 shows the circuit configuration in detail. Each line is continuously pulled up to V_{CC} through a resistive device of relatively high impedance.

This pullup is sufficient to provide the source current for a TTL high level yet can be pulled low by a standard TTL gate thus allowing the same pin to be used for both input and output. To provide fast switching times in a "0" to "1" transition a relatively low impedance device is switched in momentarily ($\approx$ 1/5 of a machine cycle) whenever a "1" is written to the line. When a "0" is written to the line a low impedance device overcomes the light pullup and provides TTL current sinking capability. Since the pulldown transistor is a low impedance device a "1" must first be written to any line which is to be used as an input. Reset initializes all lines to the high impedance "1" state.

It is important to note that the ORL and the ANL are read/write operations. When executed, the μC "reads" the port, modifies the data according to the instruction, then "writes" the data back to the port. The "writing" (essentially an OUTL instruction) enables the low impedance pull-up momentarily again even if the data was unchanged from a "1." This specifically applies to configurations that have inputs and outputs mixed together on the same port. See also section 8 in the Expanded MCS-48 System chapter.

BUS

Bus is also an 8-bit port which is a true bidirectional port with associated input and output strobes. If the bidirectional feature is not needed, Bus can serve as either a statically latched output port or non-latching input port. Input and output lines on this port cannot be mixed however.

As a static port, data is written and latched using the OUTL instruction and inputted using the INS instruction. The INS and OUTL instructions generate pulses on the corresponding $\overline{RD}$ and $\overline{WR}$ output strobe lines; however, in the static port mode they are generally not used. As a bidirectional port the MOVX instructions are used to read and write the port. A write to the port generates a pulse on the $\overline{WR}$ output line and output data is valid at the trailing edge of $\overline{WR}$. A read of the port generates a pulse on the $\overline{RD}$ output line and input data must be valid at the trailing edge of $\overline{RD}$. When not being written or read, the BUS lines are in a high impedance state. See also sections 7 and 8 in the Expanded MCS-48 System chapter.

2.5 Test and INT Inputs

Three pins serve as inputs and are testable with the conditional jump instruction. These are T0, T1, and $\overline{INT}$. These pins allow inputs to cause program branches without the necessity to load an input port into the accumulator. The T0, T1, and $\overline{INT}$ pins have other possible functions as well. See the pin description in Section 3.

2.6 Program Counter and Stack

The Program Counter is an independent counter while the Program Counter Stack is implemented suing pairs of registers in the Data Memory Array. Only 10, 11, or 12 bits of the Program Counter are used to address the 1024, 2048, or 4096 words of on-board program memory of the 8048AH, 8049AH, or 8050AH, while the most significant bits can be used for external Program Memory fetches. See Figure 5. The Program Counter is initialized to zero by activating the Reset line.

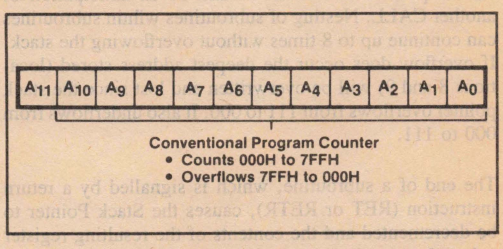


Figure 5. Program Counter

An interrupt or CALL to a subroutine causes the contents of the program counter to be stored in one of the 8 register pairs of the Program Counter Stack as shown in Figure 6. The pair to be used is determined by a 3-bit Stack Pointer which is part of the Program Status Word (PSW).

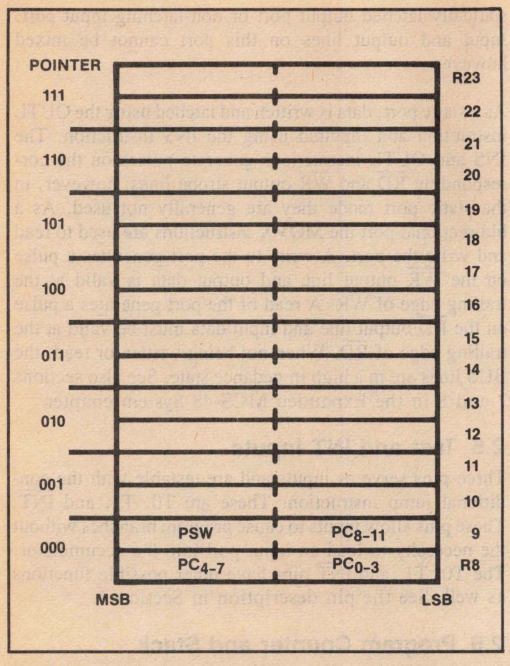


Figure 6. Program Counter Stack

Data RAM locations 8-23 are available as stack registers and are used to store the Program Counter and 4 bits of PSW as shown in Figure 6. The Stack Pointer .when initialized to 000 points to RAM locations 8 and 9. The first subroutine jump or interrupt results in the program counter contents being transferred to locations 8 and 9 of the RAM array. The stack pointer is then incremented by one to point to locations 10 and 11 in anticipation of another CALL. Nesting of subroutines wihtin subroutines can continue up to 8 times without overflowing the stack. If overflow does occur the deepest address stored (locations 8 and 9) will be overwritten and lost since the stack pointer overflows from 111 to 000. It also underflows from 000 to 111.

The end of a subroutine, which is signalled by a return instruction (RET or RETR), causes the Stack Pointer to be decremented and the contents of the resulting register pair to be transferred to the Program Counter.

2.7 Program Status Word

An 8-bit status word which can be loaded to and from the accumulator exists called the Program Status Word (PSW). Figure 7 shows the information available in the word. The Program Status Word is actually a collection of flip-flops throughout the machine which can be read or written as a whole. The ability to write to PSW allows for easy restoration of machine status after a power down sequence.

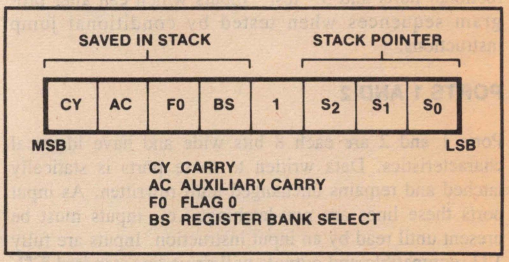


Figure 7. Program Status Word (PSW)

The upper four bits of PSW are stored in the Program Counter Stack with every call to subroutine or interrupt vector and are optionally restored upon return with the RETR instruction. The RET return instruction does not update PSW.

The PSW bit definitions are as follows:

Bits 0–2: Stack Pointer bits (S_0, S_1, S_2)

Bit 3: Not used (''1'' level when read)

Bit 4: Working Register Bank Switch Bit (BS)
0 = Bank 0
1 = Bank 1

Bit 5: Flag 0 bit (F0) user controlled flag which can be complemented or cleared, and tested with the conditional jump instruction JF0.

Bit 6: Auxiliary Carry (AC) carry bit generated by an ADD instruction and used by the decimal adjust instruction DA A.

Bit 7: Carry (CY) carry flag which indicates that the previous operation has resulted in overflow of the accumulator.

2.8 Conditional Branch Logic

The conditional branch logic within the processsor enables several conditions internal and external to the processor to be tested by the users program. By using the conditional jump instruction the conditions that are listed in Table 1 can effect a change in the sequence of the program execution.

Table 1

Device Testable	Jump Conditions (Jump On)	
	All zeros	not all zeros
Accumulator	—	1
Accumulator Bit	0	1
Carry Flag	—	1
User Flags (F0, F1)	—	1
Timer Overflow Flag	0	1
Test Inputs (T0, T1)	0	1
Interrupt Input ($\overline{INT}$)	0	—

2.9 Interrupt

An interrupt sequence is initiated by applying a low "0" level input to the INT pin. Interrupt is level triggered and active low to allow "WIRE ORing" of several interrupt sources at the input pin. Figure 8 shows the interrupt logic of the 8048AH. The Interrupt line is sampled every instruction cycle and when detected causes a "call to subroutine" at location 3 in program memory as soon as all cycles of the current instruction are complete. On 2-cycle instructions the interrupt line is sampled on the 2nd cycle only. $\overline{INT}$ must be held low for at least 3 machine cycles to ensure proper interrupt operations. As in any CALL to subroutine, the Program Counter and Program Status word are saved in the stack. For a description of this operation see the previous section, Program Counter and Stack. Program Memory location 3 usually contains an unconditional jump to an interrupt service subroutine elsewhere in program memory. The end of an interrupt service subroutine is signalled by the execution of a Return and Restore Status instruction RETR. The interrupt system is single level in that once an interrupt is detected all further interrupt requests are ignored until execution of an RETR reenables the interrupt input logic. This occurs at the beginning of the second cycle of the RETR instruction. This sequence holds true also for an internal interrupt generated by timer overflow. If an internal timer/counter generated interrupt and an external interrupt are detected at the same time, the external source will be recognized. See the following Timer/Counter section for a description of timer interrupt. If needed, a second external interrupt can be created by enabling the timer/counter interrupt, loading FFH in the Counter (ones less than terminal count), and enabling the event counter mode. A "1" to "0" transition on the T1 input will then cause an interrupt vector to location 7.

INTERRUPT TIMING

The interrupt input may be enabled or disabled under Program Control using the EN I and DIS I instructions. Interrupts are disabled by Reset and remain so until enabled by the users program. An interrupt request must be removed before the RETR instruction is executed upon return from the service routine otherwise the processor will re-enter the service routine immediately. Many peripheral devices prevent this situation by resetting their interrupt request line whenever the processor accesses (Reads or Writes) the peripherals data buffer register. If the interrupting device does not require access by the processor, one output line of the 8048AH may be designated as an "interrupt acknowledge" which is activated by the service subroutine to reset the interrupt request. The $\overline{INT}$ pin may also be tested using the conditional jump instruction JNI. This instruction may be used to detect the presence of a pending interrupt before interrupts are enabled. If interrupt is left disabled, $\overline{INT}$ may be used as another test input like T0 and T1.

2.10 Timer/Counter

The 8048AH contains a counter to aid the user in counting external events and generating accurate time delays without placing a burden on the processor for these functions. In both modes the counter operation is the same, the only difference being the source of the input to the counter. The timer/event counter is shown in Figure 9.

COUNTER

The 8-bit binary counter is presettable and readable with two MOV instructions which transfer the contents of the accumulator to the counter and vice versa. The counter content may be affected by Reset and should be initialized by software. The counter is stopped by a Reset or STOP TCNT instruction and remains stopped until started as a timer by a START T instruction or as an event counter by a START CNT instruction. Once started the counter will increment to this maximum count (FF) and overflow to zero continuing its count until stopped by a STOP TCNT instruction or Reset.

The increment from maximum count to zero (overflow) results in the setting of an overflow flag flip-flop and in the generation of an interrupt request. The state of the overflow flag is testable with the conditional jump instruction JTF. The flag is reset by executing a JTF or by Reset. The interrupt request is stored in a latch and then ORed with the external interrupt input INT. The timer interrupt may be enabled or disabled independently of external interrupt by the EN TCNT1 and DIS TCNT1 instructions. If enabled, the counter overflow will cause a subroutine call to location 7 where the timer or counter service routine may be stored.

If timer and external interrupts occur simultaneously, the external source will be recognized and the Call will be to

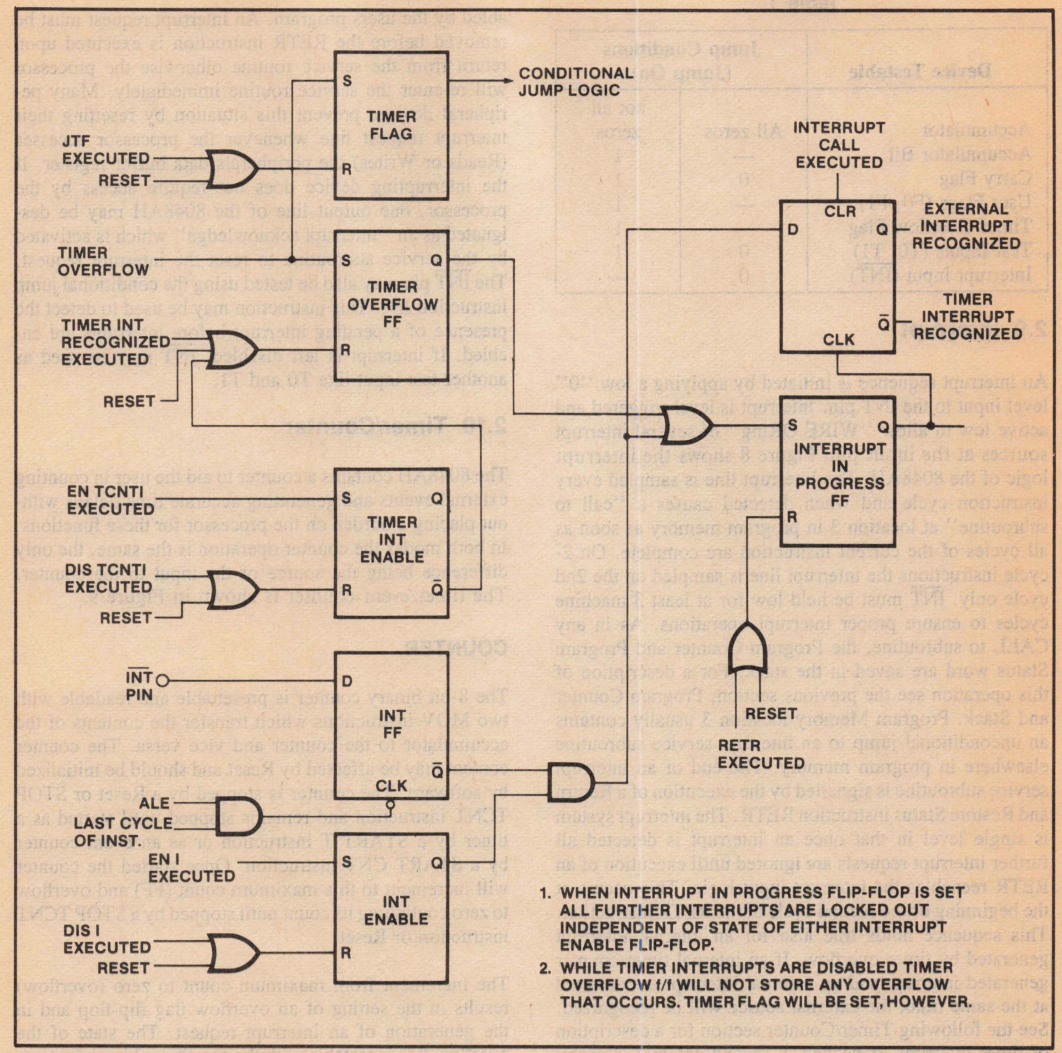


Figure 8. Interrupt Logic

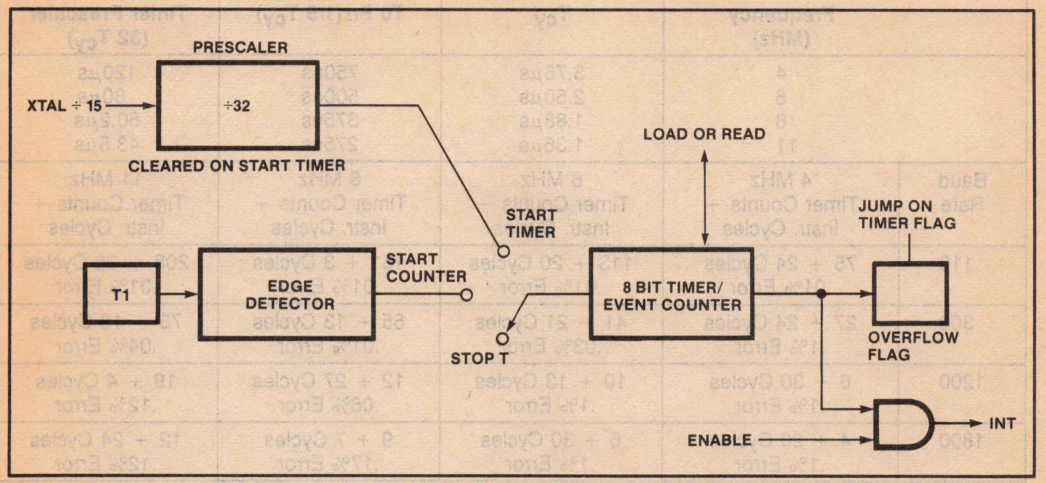


Figure 9. Timer/Event Counter

location 3. Since the timer interrupt is latched it will remain pending until the external device is serviced and immediately be recognized upon return from the service routine. The pending timer interrupt is reset by the Call to location 7 or may be removed by executing a DIS TCNT1 instruction.

AS AN EVENT COUNTER

Execution of a START CNT instruction connects the T1 input pin to the counter input and enables the counter. The T1 input is sampled at the beginning of state 3 or in later MCS-48 devices in state time 4. Subsequent high to low transitions on T1 will cause the counter to increment. T1 must be held low for at least 1 machine cycle to insure it won't be missed. The maximum rate at which the counter may be incremented is once per three instruction cycles (every 5.7 μsec when using an 8 MHz crystal) — there is no minimum frequency. T1 input must remain high for at least 1/5 machine cycle after each transition.

AS A TIMER

Eexcution of a START T instruction connects an internal clock to the counter input and enables the counter. The internal clock is derived bypassing the basic machine cycle clock through a ÷32 prescaler. The prescaler is reset during the START T instruction. The resulting clock increments the counter every 32 machine cycles. Various delays from 1 to 256 counts can be obtained by presetting the counter and detecting overflow. Times longer than 256 counts may be achieved by accumulating multiple overflows in a register under software control. For time resolution less than 1 count an external clock can be applied to the T1 input and the counter operated in the event counter mode. ALE divided by 3 or more can serve as this external clock. Very small delays or "fine tuning" of larger delays can be easily accomplished by software delay loops.

Often a serial link is desirable in an MCS-48 family member. Table 2 lists the timer counts and cycles needed for a specific baud rate given a crystal frequency.

2.11 Clock and Timing Circuits

Timing generation for the 8048AH is completely selfcontained with the exeception of a frequency reference which can be XTAL, ceramic resonator, or external clock source. The Clock and Timing circuitry can be divided into the following functional blocks.

OSCILLATOR

The on-board oscillator is a high gain parallel resonant circuit with a frequency range of 1 to 11 MHz. The X1 external pin is the input to the amplifier stage while X2 is the output. A crystal or ceramic resonator connected between X1 and X2 provides the feedback and phase shift required for oscillation. If an accurate frequency reference is not required, ceramic resonator may be used in place of the crystal.

For accurate clocking, a crystal should be used. An externally generated clock may also be applied to X1–X2 as the frequency source. See the data sheet for more information.

Table 2. Baud Rate Generation

	Frequency (MHz)	T_{cy}	T0 Prr(1/5 T_{cy})	Timer Prescaler (32 T_{cy})
	4	3.75μs	750ns	120μs
	6	2.50μs	500ns	80μs
	8	1.88μs	375ns	60.2μs
	11	1.36μs	275ns	43.5μs
Baud Rate	4 MHz Timer Counts + Instr. Cycles	6 MHz Timer Counts + Instr. Cycles	8 MHz Timer Counts + Instr. Cycles	11 MHz Timer Counts + Instr. Cycles
110	75 + 24 Cycles .01% Error	113 + 20 Cycles .01% Error	151 + 3 Cycles .01% Error	208 + 28 Cycles .01% Error
300	27 + 24 Cycles .1% Error	41 + 21 Cycles .03% Error	55 + 13 Cycles .01% Error	76 + 18 Cycles .04% Error
1200	6 + 30 Cycles .1% Error	10 + 13 Cycles .1% Error	12 + 27 Cycles .06% Error	19 + 4 Cycles .12% Error
1800	4 + 20 Cycles .1% Error	6 + 30 Cycles .1% Error	9 + 7 Cycles .17% Error	12 + 24 Cycles .12% Error
2400	3 + 15 Cycles .1% Error	5 + 6 Cycles .4% Error	6 + 24 Cycles .29% Error	9 + 18 Cycles .12% Error
4800	1 + 23 Cycles 1.0% Error	2 + 19 Cycles .4% Error	3 + 14 Cycles .74% Error	4 + 25 Cycles .12% Error

STATE COUNTER

The output of the oscillator is divided by 3 in the State Counter to create a clock which defines the state times of the machine (CLK). CLK can be made available on the external pin T0 by executing an ENTO CLK instruction. The output of CLK on T0 is disabled by Reset of the processor.

CYCLE COUNTER

CLK is then divided by 5 in the Cycle Counter to provide a clock which defines a machine cycle consisting of 5 machine states as shown in Figure 10. Figure 11 shows the different internal operations as divided into the machine states. This clock is called Address Latch Enable (ALE) because of its function in MCS-48 systems with external memory. It is provided continuously on the ALE output pin.

2.12 Reset

The reset input provides a means for initialization for the processor. This Schmitt-trigger input has an internal pull-up device which in combination with an external 1 μ fd capacitor provides an internal reset pulse of sufficient length to guarantee all circuitry is reset, as shown in Figure 12. If the reset pulse is generated externally the RESET pin must be held low for at least 10 milliseconds after the power supply is within tolerance. Only 5 machine cycles (6.8 μs @ 11 MHz) are required if power is already on and the oscillator has stabilized. ALE and PSEN (if EA = 1) are active while in Reset.

Reset performs the following functions:

1) Sets program counter to zero.

2) Sets stack pointer to zero.

3) Selects register bank 0.

4) Selects memory bank 0.

5) Sets BUS to high impedance state (except when EA = 5V).

6) Sets Ports 1 and 2 to input mode.

7) Disables interrupts (timer and external).

8) Stops timer.

9) Clears timer flag.

10) Clears F0 and F1.

11) Disables clock output from T0.

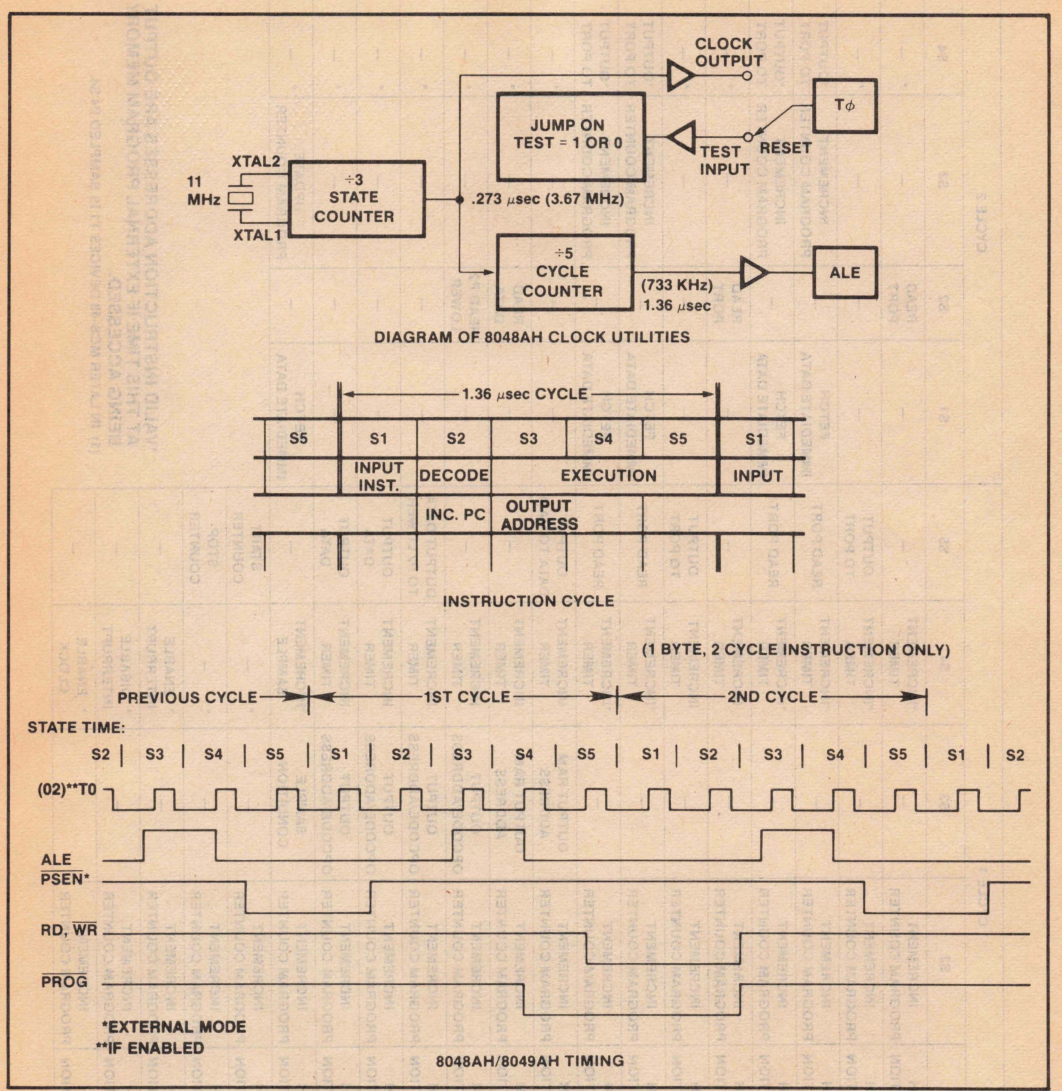


Figure 10. MCS®-48 Timing Generation and Cycle Timing

2.13 Single-Step

This feature, as pictured in Figure 13, provides the user with a debug capability in that the processor can be stepped through the program one instruction at a time. While stopped, the address of the next instruction to be fetched is available concurrently on BUS and the lower half of Port 2. The user can therefore follow the program through each of the instruction steps. A timing diagram, showing the interaction between output ALE and input $\overline{SS}$, is shown. The BUS buffer contents are lost during single step; however, a latch may be added to reestablish the lost I/O capability if needed. Data is valid at the leading edge of ALE.

	CYCLE 1					CYCLE 2				
INSTRUCTION	S1	S2	S3	S4	S5	S1	S2	S3	S4	S5
IN A,P	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	—	*INCREMENT TIMER	—	—	READ PORT	—	* —	—
OUTL P,A	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	—	*INCREMENT TIMER	OUTPUT TO PORT	—	—	—	* —	—
ANL P, = DATA	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	—	*INCREMENT TIMER	READ PORT	FETCH IMMEDIATE DATA	—	INCREMENT PROGRAM COUNTER	*OUTPUT TO PORT	—
ORL P, = DATA	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	—	*INCREMENT TIMER	READ PORT	FETCH IMMEDIATE DATA	—	INCREMENT PROGRAM COUNTER	*OUTPUT TO PORT	—
INS A, BUS	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	—	INCREMENT TIMER	—	—	READ PORT	—	* —	—
OUTL BUS, A	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	—	INCREMENT TIMER	OUTPUT TO PORT	—	—	—	* —	—
ANL BUS, = DATA	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	—	*INCREMENT TIMER	READ PORT	FETCH IMMEDIATE DATA	—	INCREMENT PROGRAM COUNTER	*OUTPUT TO PORT	—
ORL BUS, = DATA	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	—	*INCREMENT TIMER	READ PORT	FETCH IMMEDIATE DATA	—	INCREMENT PROGRAM COUNTER	*OUTPUT TO PORT	—
MOVX @ R,A	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	OUTPUT RAM ADDRESS	INCREMENT TIMER	OUTPUT DATA TO RAM	—	—	—	* —	—
MOVX A,@R	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	OUTPUT RAM ADDRESS	INCREMENT TIMER	—	—	READ DATA	—	* —	—
MOVD A,P_i	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	OUTPUT OPCODE/ADDRESS	INCREMENT TIMER	—	—	READ P2 LOWER	—	* —	—
MOVD P_i,A	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	OUTPUT OPCODE/ADDRESS	INCREMENT TIMER	OUTPUT DATA TO P2 LOWER	—	—	—	* —	—
ANLD P,A	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	OUTPUT OPCODE/ADDRESS	INCREMENT TIMER	OUTPUT DATA	—	—	—	* —	—
ORLD P,A	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	OUTPUT OPCODE/ADDRESS	INCREMENT TIMER	OUTPUT DATA	—	—	—	* —	—
J(CONDITIONAL)	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	SAMPLE CONDITION	*INCREMENT SAMPLE	—	FETCH IMMEDIATE DATA	—	UPDATE PROGRAM COUNTER	* —	—
STRT T STRT CNT	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	—	* —	START COUNTER					
STOP TCNT	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	—	* —	STOP COUNTER					
ENI	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	—	* ENABLE INTERRUPT	—					
DIS I	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	—	* DISABLE INTERRUPT	—					
ENTO CLK	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER		* ENABLE CLOCK	—					

*VALID INSTRUCTION ADDRESSES ARE OUTPUT AT THIS TIME IF EXTERNAL PROGRAM MEMORY IS BEING ACCESSED.

(1) IN LATER MCS-48 DEVICES T1 IS SAMPLED IN S4.

Figure 11. 8048AH/8049AH Instruction Timing Diagram

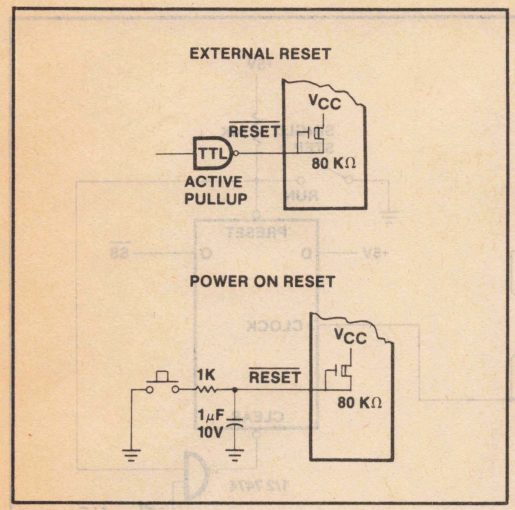


Figure 12.

TIMING

The 8048AH operates in a single-step mode as follows:

1) The processor is requested to stop by applying a low level on $\overline{SS}$.
2) The processor responds by stopping during the address fetch portion of the next instruction. If a double cycle instruction is in progress when the single step command is received, both cycles will be completed before stopping.
3) The processor acknowledges it has entered the stopped state by raising ALE high. In this state (which can be maintained indefinitely) the address of the next instruction to be fetched is present on BUS and the lower half of port 2.
4) $\overline{SS}$ is then raised high to bring the processor out of the stopped mode allowing it to fetch the next instruction. The exit from stop is indicated by the processor bringing ALE low.
5) To stop the processor at the next instruction $\overline{SS}$ must be brought low again soon after ALE goes low. If $\overline{SS}$ is left high the processor remains in a "Run" mode.

A diagram for implementing the single-step function of the 8748H is shown in Figure 13. D-type flip-flop with preset and clear is used to generate $\overline{SS}$. In the run mode SS is held high by keeping the flip-flop preset (preset has precedence over the clear input). To enter single step, preset is removed allowing ALE to bring $\overline{SS}$ low via the clear input. ALE should be buffered since the clear input of an SN7474 is the equivalent of 3 TTL loads. The processor is now in the stopped state. The next instruction is initiated by clocking a "1" into the flip-flop. This "1" will not appear on $\overline{SS}$ unless ALE is high removing clear from the flip-flop. In response to $\overline{SS}$ going high the processor begins an instruction fetch which brings ALE low resetting $\overline{SS}$ through the clear input and causing the processor to again enter the stopped state.

2.14 Power Down Mode (8048AH, 8049AH, 8050AH, 8039AHL, 8035AHL, 8040AHL)

Extra circuitry has been added to the 8048AH/8049AH/8050AH ROM version to allow power to be removed from all but the data RAM array for low power standby operation. In the power down mode the contents of data RAM can be maintained while drawing typically 10% to 15% of normal operating power requirements.

V_{CC} serves as the 5V supply pin for the bulk of circuitry while the V_{DD} pin supplies only the RAM array. In normal operation both pins are a 5V while in standby, V_{CC} is at ground and V_{DD} is maintained at its standby value. Applying Reset to the processor through the $\overline{RESET}$ pin inhibits any access to the RAM by the processor and guarantees that RAM cannot be inadvertently altered as power is removed from V_{CC}.

A typical power down sequence (Figure 14) occurs as follows:

1) Imminent power supply failure is detected by user defined circuitry. Signal must be early enough to allow 8048AH to save all necessary data before V_{CC} falls below normal operating limits.
2) Power fail signal is used to interrupt processor and vector it to a power fail service routine.
3) Power fail routine saves all important data and machine status in the internal data RAM array. Routine may also initiate transfer of backup supply to the V_{DD} pin and indicate to external circuitry that power fail routine is complete.
4) Reset is applied to guarantee data will not be altered as the power supply falls out of limits. Reset must be held low until V_{CC} is at ground level.

Recovery from the Power Down mode can occur as any other power-on sequence with an external capacitor on the Reset input providing the necessary delay. See the previous section on Reset.

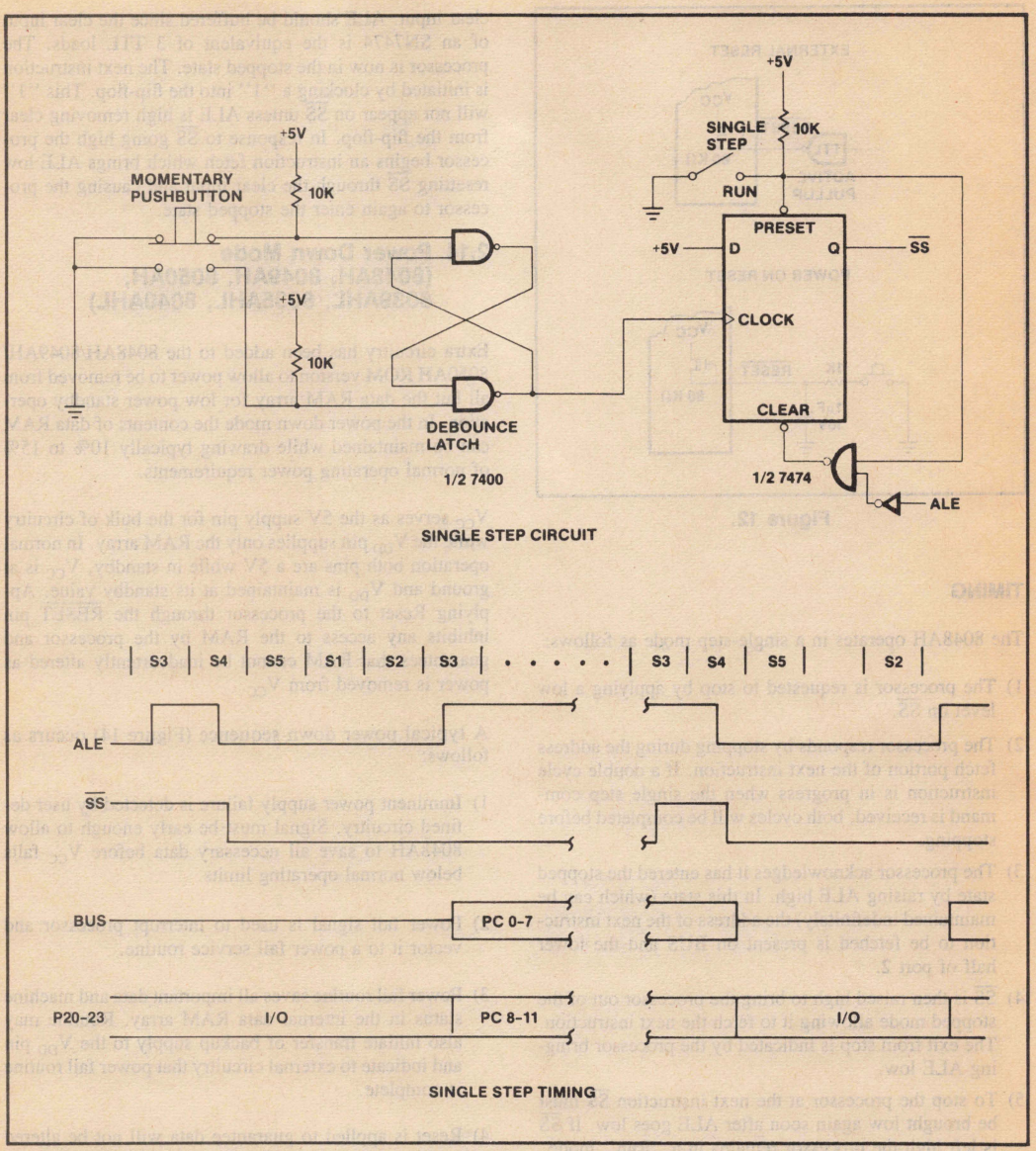


Figure 13. Single Step Operation

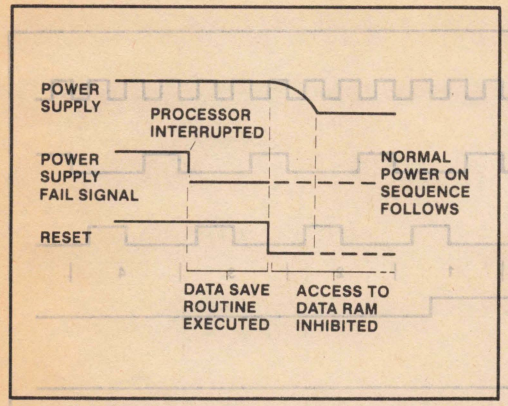


Figure 14. Power Down Sequence

2.15 External Access Mode

Normally the first 1K (8048AH), 2K (8049AH), or 4K (8050AH) words of program memory are automatically fetched from internal ROM or EPROM. The EA input pin however allows the user to effectively disable internal program memory by forcing all program memory fetches to reference external memory. The following chapter explains how access to external program memory is accomplished.

The External Access mode is very useful in system test and debug because it allows the user to disable his internal applications program and substitute an external program of his choice — a diagnostic routine for instance. In addition, the date sheet shows how internal program memory can be read externally, independent of the processor. A "1" level on EA initiates the external accesss mode. For proper operation, Reset should be applied while the EA input is changed.

2.16 Sync Mode

The 8048AH, 8049AH, 8050AH has incorporated a new SYNC mode. The Sync mode is provided to ease the design of multiple controller circuits by allowing the designer to force the device into known phase and state time. The SYNC mode may also be utilized by automatic test equipment (ATE) for quick, easy, and efficient synchronizing between the tester and the DUT (device under test).

SYNC mode is enabled when SS′ pin is raised to high voltage level of + 12 volts. To begin synchronization, T0 is raised to 5 volts at least four clocks cycles after SS′. T0 must be high for at least four X1 clock cycles to fully reset the prescaler and time state generators. T0 may then be brought down with the rising edge of X1. Two clock cycles later, with the rising edge of X1, the device enters into Time State 1, Phase 1, SS′ is then brought down to 5 volts 4 clocks later after T0. RESET′ is allowed to go high 5 tCY (75 clocks) later for normal execution of code. See Figure 15.

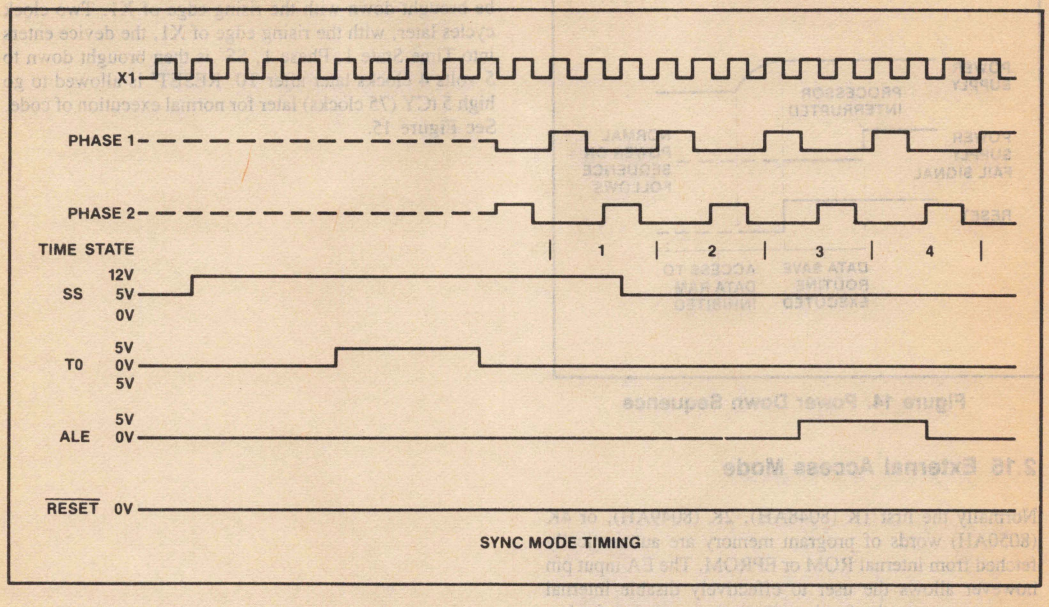


Figure 15. Sync Mode Timing

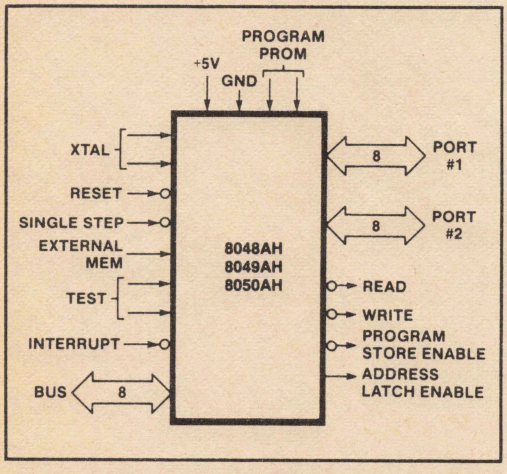


Figure 16. 8048AH and 8049AH Logic Symbol

3.0 PIN DESCRIPTION

The MCS-48 processors are packaged in 40 pin Dual In-Line Packages (DIP's). Table 3 is a summary of the functions of each pin. Figure 16 is the logic symbol for the 8048AH product family. Where it exists, the second paragraph describes each pin's function in an expanded MCS-48 system. Unless otherwise specified, each input is TTL compatible and each output will drive one standard TTL load.

Table 3. Pin Description

Designation	Pin Number*	Function
V_{SS}	20	Circuit GND potential
V_{DD}	26	Programming power supply; 21V during program for the 8748H/8749H; +5V during operation for both ROM and EPROM. Low power standby pin in 8048AH and 8049AH/8050AH ROM versions.
V_{CC}	40	Main power supply; +5V during operation and during 8748H and 8749H programming.
PROG	25	Program pulse; +18V input pin during 8748H/8749H programming. Output strobe for 8243 I/O expander.
P10–P17 (Port 1)	27-34	8-bit quasi-bidirectional port. (Internal Pullup ≈ 50KΩ)
P20–P27 (Port 2)	21–24 35–38	8-bit quasi-bidirectional port. (Internal Pullup ≈ 50KΩ) P20–P23 contain the four high order program counter bits during an external program memory fetch and serve as a 4-bit I/O expander bus for 8243.
D0–D7 (BUS)	12–19	True bidirectional port which can be written or read synchronously using the $\overline{RD}$, $\overline{WR}$ strobes. The port can also be statically latched. Contains the 8 low order program counter bits during an external program memory fetch, and receives the addressed instruction under the control of $\overline{PSEN}$. Also contains the address and data during an external RAM data store instruction, under control of ALE, $\overline{RD}$, and $\overline{WR}$.
T0	1	Input pin testable using the conditional transfer instructions JT0 and JNT0. T0 can be designated as a clock output using ENTO CLK instruction. T0 is also used during programming and sync mode.
T1	39	Input pin testable using the JT1, and JNT1 instructions. Can be designated the event counter input using the STRT CNT instruction. (See Section 2.10).
$\overline{INT}$	6	Interrupt input. Initiates an interrupt if interrupt is enabled. Interrupt is disabled after a reset. (Active low) Interrupt must remain low for at least 3 machine cycles to ensure proper operation.
$\overline{RD}$	8	Output strobe activated during a BUS read. Can be used to enable data onto the BUS from an external device. (Active low) Used as a Read Strobe to External Data Memory.
$\overline{RESET}$	4	Input which is used to initialize the processor. Also used during EPROM programming and verification. (Active low) (Internal pullup ≈ 80K Ω)
$\overline{WR}$	10	Output strobe during a BUS write. (Active low) Used as write strobe to external data memory.
ALE	11	Address Latch Enable. This signal occurs once during each cycle and is useful as a clock output. The negative edge of ALE strobes address into external data and program memory.

Table 3. Pin Description (Continued)

Designation	Pin Number*	Function
$\overline{\text{PSEN}}$	9	Program Store Enable. This output occurs only during a fetch to external program memory. (Active low)
$\overline{\text{SS}}$	5	Single step input can be used in conjunction with ALE to "single step" the processor through each instruction. (Active low) (Internal pullup ≈ 300KΩ) +12V for sync modes (See 2.16).
EA	7	External Access input which forces all program memory fetches to reference external memory. Useful for emulation and debug, and essential for testing and program verification. (Active high) +12V for 8048AH/8049AH/8050AH program verification and +18V for 8748H/8749H program verification (Internal pullup ≈ 10MΩ on 8048AH/8049AH/8035AHL/8039AHL/8050AH/8040AHL)
XTAL1	2	One side of crystal input for internal oscillator. Also input for external source.
XTAL2	3	Other side of crystal/external source input.

*Unless otherwise stated, inputs do not have internal pullup resistors. 8048AH, 8748H, 8049AH, 8050AH, 8040AHL

4.0 PROGRAMMING, VERIFYING AND ERASING EPROM

The internal Program Memory of the 8748H and the 8749H may be erased and reprogrammed by the user as explained in the following sections. See also the 8748H and 8749H data sheets.

4.1 Programming/Verification

In brief, the programming process consists of: activating the program mode, applying an address, latching the address, applying data, and applying a programming pulse. This programming algorithm applies to both the 8748H and 8749H. Each word is programmed completely before moving on to the next and is followed by a verification step. The following is a list of the pins used for programming and a descsription of their functions:

Pin	Function
XTAL 1	Clock Input (3 to 4 MHz)
Reset	Initialization and Address Latching
Test 0	Selection of Program (0V) or Verify (5V) Mode
EA	Activation of Program/Verify Modes
BUS	Address and Data Input Data Output During Verify
P20-1	Address Input for 8748H
P20-2	Address Input for 8749H
V_{DD}	Programming Power Supply
PROG	Program Pulse Input
P10–P11	Tied to ground (8749H only)

8748H AND 8749H ERASURE CHARACTERISTICS

The erasure characteristics of the 8748H and 8749H are such that erasure begins to occur when exposed to light with wavelengths shorter than approximately 4000 Angstroms (A). It should be noted that sunlight and certain types of fluorescent lamps have wavelengths in the 3000–4000A range. Data show that constant exposure to room level fluorescent lighting could erase the typical 8748H and 8749H in approximately 3 years while it would take approximately 1 week to cause erasure when exposed to direct sunlight. If the 8748H or 8749H is to be exposed to these types of lighting conditions for extended periods of time, opaque labels should be placed over the 8748H window to prevent unintentional erasure.

When erased, bits of the 8748H and 8749H Program Memory are in the logic "0" state.

The recommended erasure procedure for the 8748H and 8749H is exposure to shortwave ultraviolet light which has a wavelength of 2537 Angstroms (A). The integrated dose (i.e., UV intensity X exposure time) for erasure should be a minimum of 15W-sec/cm^2. The erasure time with this dosage is approximately 15 to 20 minutes using an ultraviolet lamp with a 12000μW/cm^2 power rating. The 8748H and 8749H should be placed within one inch from the lamp tubes during erasure. Some lamps have a filter in their tubes and this filter should be removed before erasure.

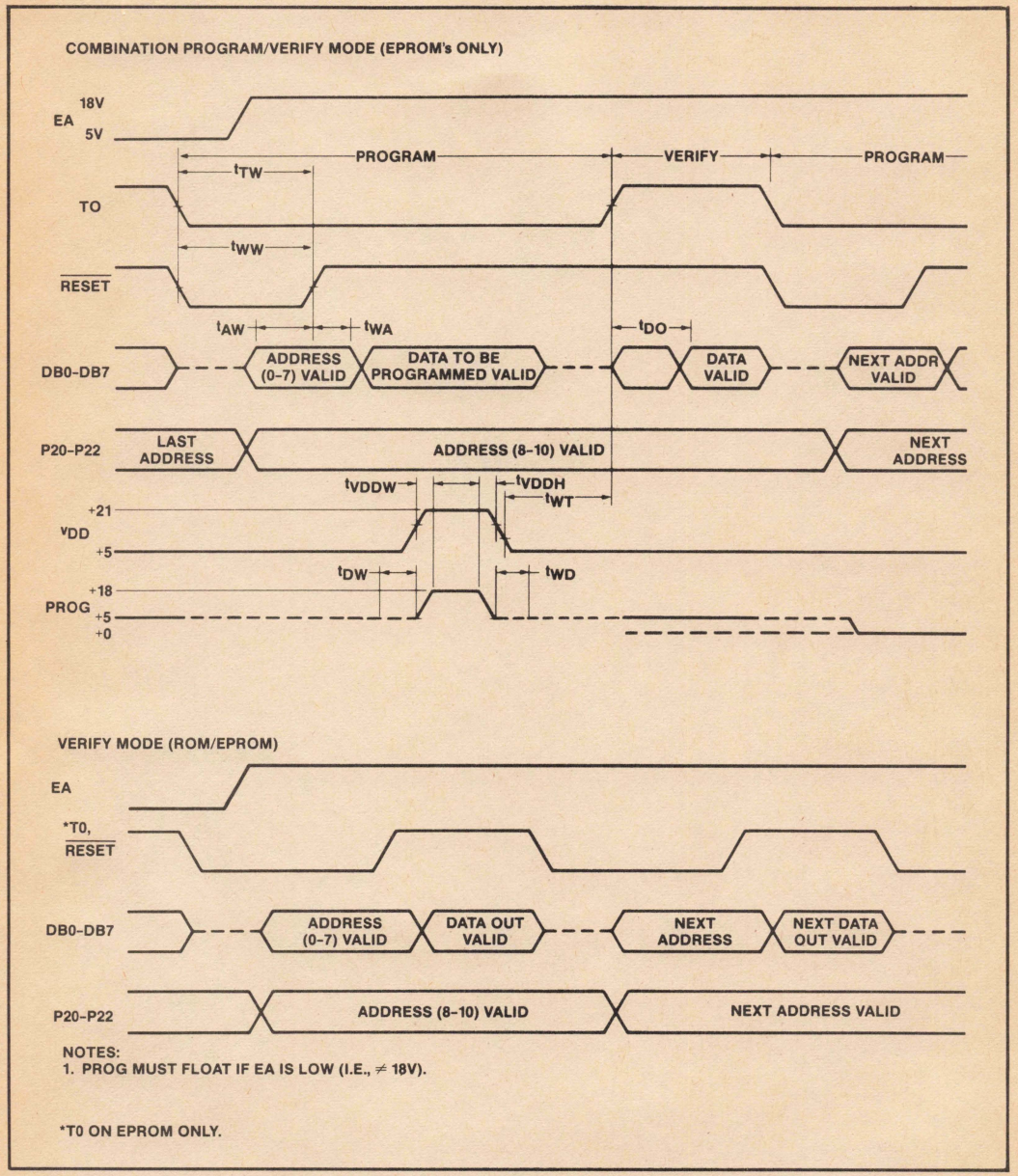


Figure 17. Program/Verify Sequence for 8749H/8748H

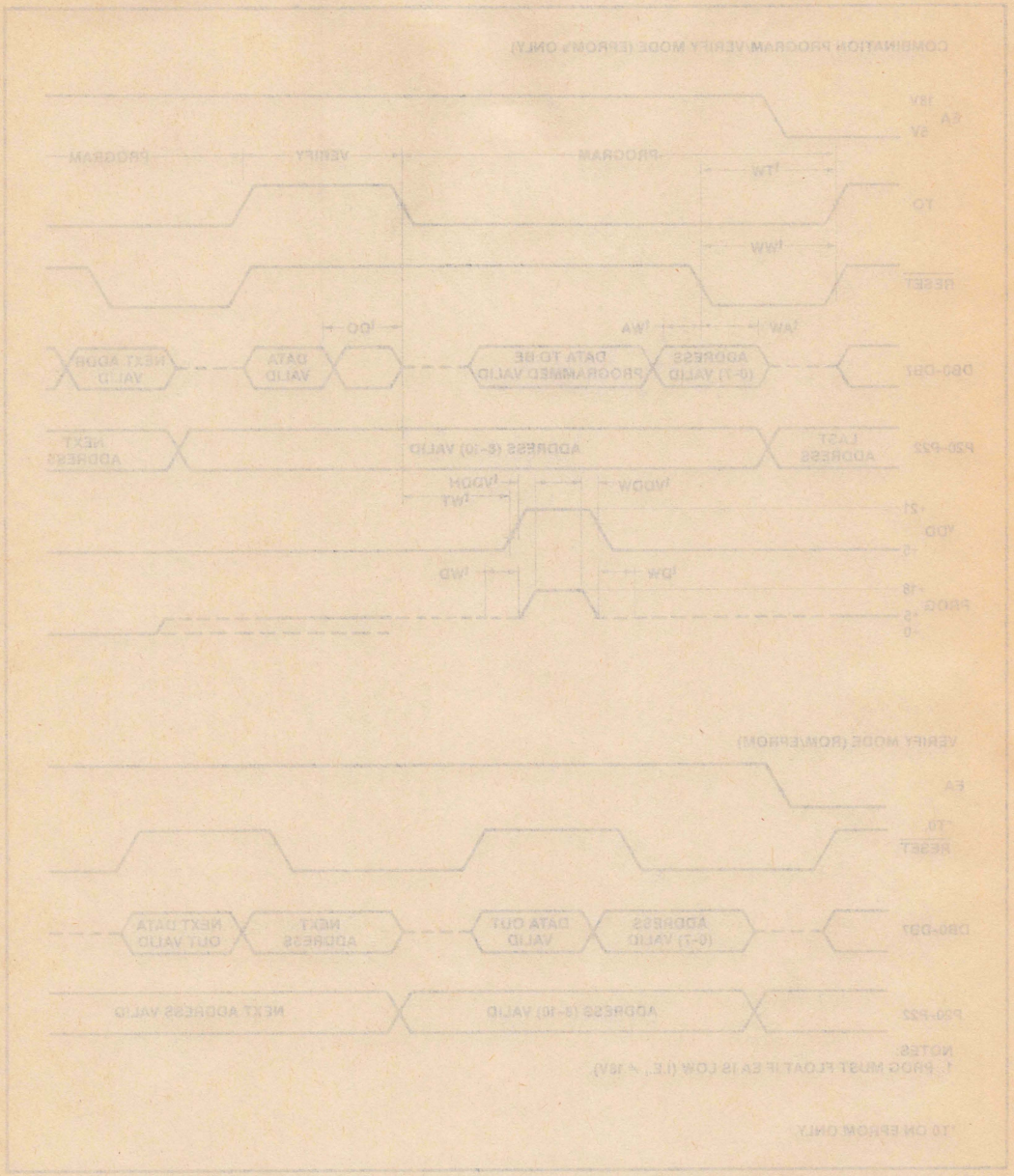


Figure 17. Program/Verify Sequence for 8742H/8748H

MCS®-48 Expanded System

2

EXPANDED MCS®-48 SYSTEM

1.0 INTRODUCTION

If the capabilities resident on the single-chip 8048AH/8748H/8035AHL/8049AH/8749H/8039AHL are not sufficient for your system requirements, special on-board circuitry allows the addition of a wide variety of external memory, I/O, or special peripherals you may require. The processors can be directly and simply expanded in the following areas:

- Program Memory to 4K words
- Data Memory to 320 words (384 words with 8049AH)
- I/O by unlimited amount
- Special Functions using 8080/8085AH peripherals

By using bank switching techniques, maximum capability is essentially unlimited. Bank switching is discussed later in the chapter. Expansion is accomplished in two ways:

1) Expander I/O — A special I/O Expander circuit, the 8243, provides for the addition of four 4-bit Input/Output ports with the sacrifice of only the lower half (4-bits) of port 2 for inter-device communication. Multiple 8243's may be added to this 4-bit bus by generating the required "chip select" lines.
2) Standard 8085 Bus — One port of the 8048AH/8049AH is like the 8-bit bidirectional data bus of the 8085 microcomputer system allowing interface to the numerous standard memories and peripherals of the MCS®-80/85 microcomputer family.

MCS-48 systems can be configured using either or both of these expansion features to optimize system capabilities to the application.

Both expander devices and standard memories and peripherals can be added in virtually any number and combination required.

2.0 EXPANSION OF PROGRAM MEMORY

Program Memory is expanded beyond the resident 1K or 2K words by using the 8085 BUS feature of the MCS®-48. All program memory fetches from the addresses less than 1024 on the 8048AH and less than 2048 on the 8049AH occur internally with no external signals being generated (except ALE which is always present). At address 1024 on the 8048AH, the processor automatically initiates external program memory fetches.

2.1 Instruction Fetch Cycle (External)

As shown in Figure 1, for all instruction fetches from addresses of 1024 (2048) or greater, the following will occur:

1) The contents of the 12-bit program counter will be output on BUS and the lower half of port 2.
2) Address Latch Enable (ALE) will indicate the time at which address is valid. The trailing edge of ALE is used to latch the address externally.
3) Program Store Enable ($\overline{PSEN}$) indicates that an external instruction fetch is in progress and serves to enable the external memory device.
4) BUS reverts to input (floating) mode and the processor accepts its 8-bit contents as an instruction word.

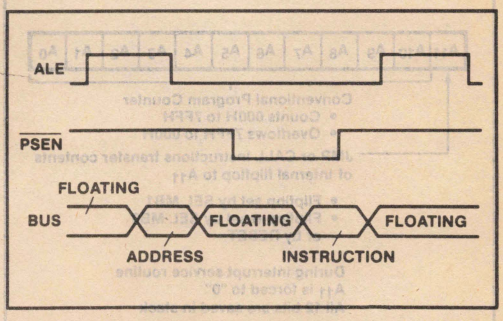


Figure 1. Instruction Fetch from External Program Memory

All instruction fetches, including internal addresses, can be forced to be external by activating the EA pin of the 8048AH/8049AH/8050AH. The 8035AHL/8039AHL/8040AHL processors without program memory always operate in the external program memory mode (EA = 5V).

2.2 Extended Program Memory Addressing (Beyond 2K)

For programs of 2K words or less, the 8048AH/8049AH addresses program memory in the conventional manner. Addresses beyond 2047 can be reached by executing a program memory bank switch instruction (SEL MB0, SEL MB1) followed by a branch instruction (JMP or CALL). The bank switch feature extends the range of branch instructions beyond their normal 2K range and at the same time prevents the user from inadvertently crossing the 2K boundary.

PROGRAM MEMORY BANK SWITCH

The switching of 2K program memory banks is accomplished by directly setting or resetting the most significant bit of the program counter (bit 11); see Figure 2. Bit 11 is not altered by normal incrementing of the program counter but is loaded with the contents of a special flip-flop each time a JMP or CALL instruction is executed. This special flip-flop is set by executing an SEL MB1

instruction and reset by SEL MB0. Therefore, the SEL MB instruction may be executed at any time prior to the actual bank switch which occurs during the next branch instruction encountered. Since all twelve bits of the program counter, including bit 11, are stored in the stack, when a Call is executed, the user may jump to subroutines across the 2K boundary and the proper bank will be restored upon return. However, the bank switch flip-flop will not be altered on return.

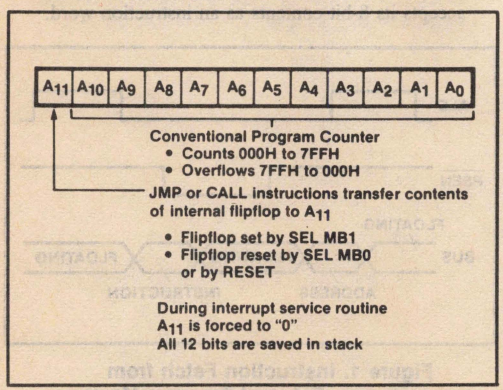


Figure 2. Program Counter

INTERRUPT ROUTINES

Interrupts always vector the program counter to location 3 or 7 in the first 2K bank, and bit 11 of the program counter is held at "0" during the interrupt service routine. The end of the service routine is signalled by the execution of an RETR instruction. Interrupt service routines should therefore be contained entirely in the lower 2K words of program memory. The execution of a SEL MB0 or SEL MB1 instruction within an interrupt routine is not recommended since it will not alter PC11 while in the routine, but will change the internal flip-flop.

2.3 Restoring I/O Port Information

Although the lower half of Port 2 is used to output the four most significant bits of address during an external program memory fetch, the I/O information is still outputed during certain portions of each machine cycle. I/O information is always present on Port 2's lower 4 bits at the rising edge of ALE and can be sampled or latched at this time.

2.4 Expansion Examples

Shown in Figure 3 is the addition of 2K words of program memory using an 2716A 2K x 8 ROM to give a total of 3K words of program memory. In this case no chip select decoding is required and $\overline{PSEN}$ enables the memory directly through the chip select input. If the system requires only 2K of program memory, the same configuration can be used with an 8035AHL substituted for the 8048AH. The 8049AH would provide 4K of program memory with the same configuration.

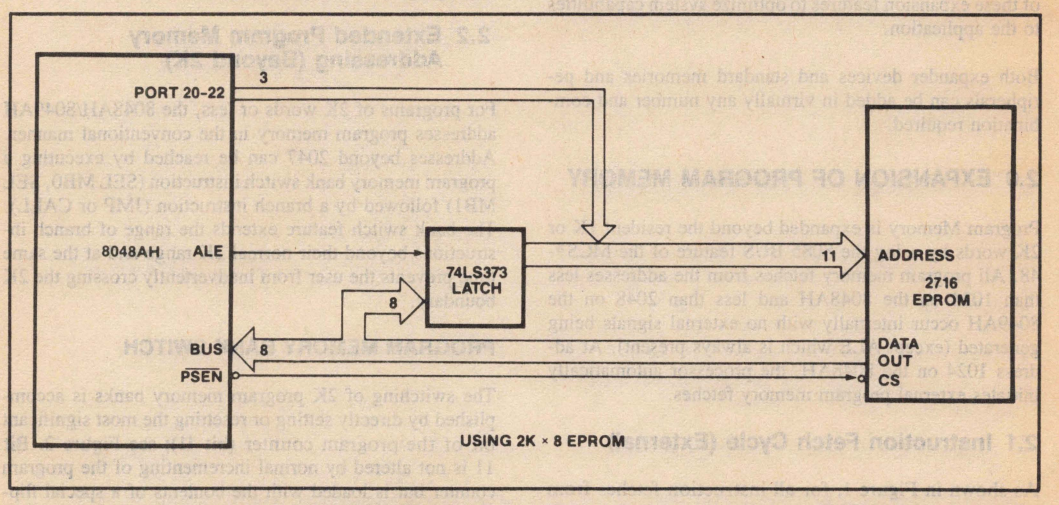


Figure 3. Expanding MCS®-48 Program Memory Using Standard Memory Products

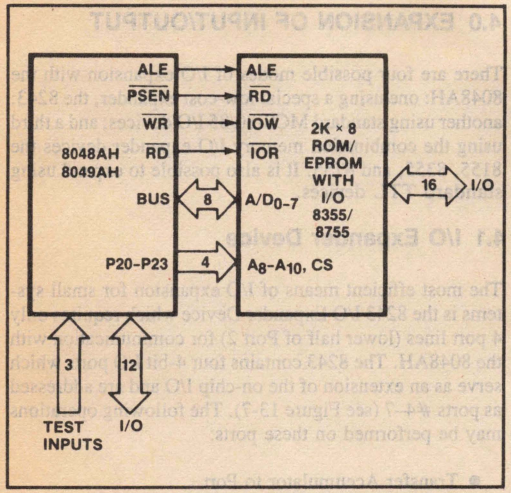


Figure 4. External Program Memory Interface

Figure 4 shows how the 8755/8355 EPROM/ROM with I/O interfaces directly to the 8048AH without the need for an address latch. The 8755/8355 contains an internal 8-bit address latch eliminating the need for an 8212 latch. In addition to a 2K x 8 program memory, the 8755/8355 also contains 16 I/O lines addressable as two 8-bit ports. These ports are addressed as external RAM; therefore the $\overline{RD}$ and $\overline{WR}$ outputs of the 8048AH are required. See the following section on data memory expansion for more detail. The subsequent section on I/O expansion explains the operation of the 16 I/O lines.

3.0 EXPANSION OF DATA MEMORY

Data Memory is expanded beyond the resident 64 words by using the 8085AH type bus feature of the MCS®-48.

3.1 Read/Write Cycle

All address and data is transferred over the 8 lines of BUS. As shown in Figure 5, a read or write cycle occurs as follows:

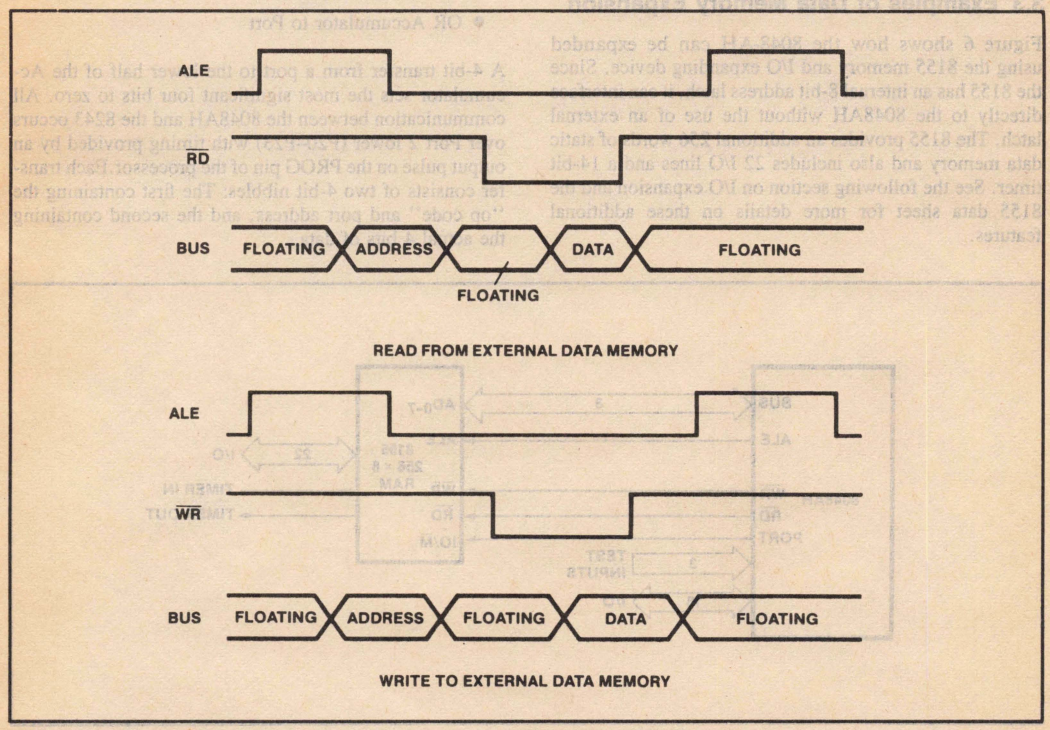


Figure 5. External Data Memory Timings

1) The contents of register R0 or R1 is outputed on BUS.
2) Address Latch Enable (ALE) indicates addresss is valid. The trailing edge of ALE is used to latch the address externally.
3) A read ($\overline{RD}$) or write ($\overline{WR}$) pulse on the corresponding output pins of the 8048AH indicates the type of data memory access in progress. Output data is valid at the trailing edge of $\overline{WR}$ and input data must be valid at the trailing edge of $\overline{RD}$.
4) Dat (8 bits) is transferred in or out over BUS.

3.2 Addressing External Data Memory

External Data Memory is accessed with its own two-cycle move instructions. MOVXA, @R and MOVX@R, A, which transfer 8 bits of data between the accumulator and the external memory location addressed by the contents of one of the RAM Pointer Registers R0 and R1. This allows 256 locations to be addressed in addition to the resident locations. Additional pages may be added by "bank switching" with extra output lines of the 8048AH.

3.3 Examples of Data Memory Expansion

Figure 6 shows how the 8048-AH can be expanded using the 8155 memory and I/O expanding device. Since the 8155 has an internal 8-bit address latch, it can interface directly to the 8048AH without the use of an external latch. The 8155 provides an additional 256 words of static data memory and also includes 22 I/O lines and a 14-bit timer. See the following section on I/O expansion and the 8155 data sheet for more details on these additional features.

4.0 EXPANSION OF INPUT/OUTPUT

There are four possible modes of I/O expansion with the 8048AH: one using a special low-cost expander, the 8243; another using standard MCS-80/85 I/O devices; and a third using the combination memory I/O expander devices the 8155, 8355, and 8755. It is also possible to expand using standard TTL devices.

4.1 I/O Expander Device

The most efficient means of I/O expansion for small systems is the 8243 I/O Expander Device which requires only 4 port lines (lower half of Port 2) for communication with the 8048AH. The 8243 contains four 4-bit I/O ports which serve as an extension of the on-chip I/O and are addressed as ports #4–7 (see Figure 13-7). The following operations may be performed on these ports:

- Transfer Accumulator to Port
- Transfer Port to Accumulator
- AND Accumulator to Port
- OR Accumulator to Port

A 4-bit transfer from a port to the lower half of the Accumulator sets the most significant four bits to zero. All communication between the 8048AH and the 8243 occurs over Port 2 lower (P20–P23) with timing provided by an output pulse on the PROG pin of the processor. Each transfer consists of two 4-bit nibbles: The first containing the "op code" and port address, and the second containing the actual 4 bits of data.

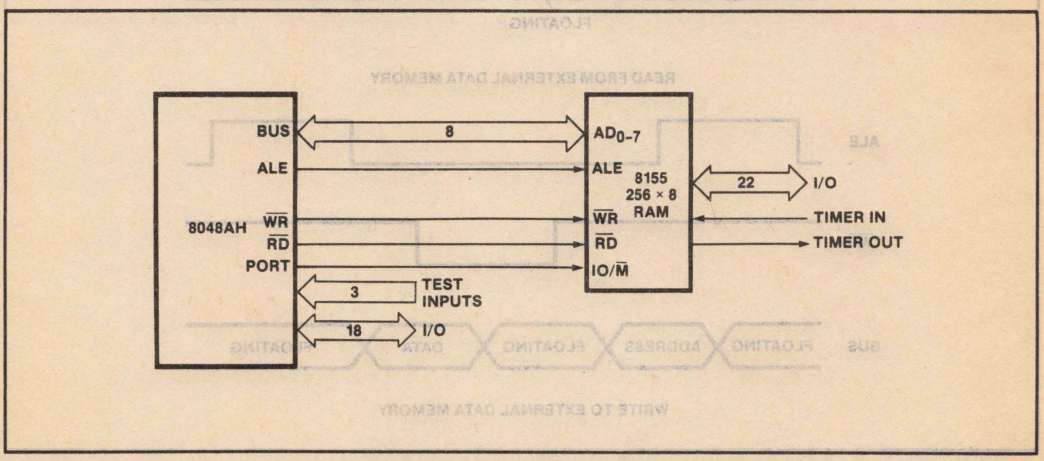


Figure 6. 8048AH Interface to 256 x 8 Standard Memories

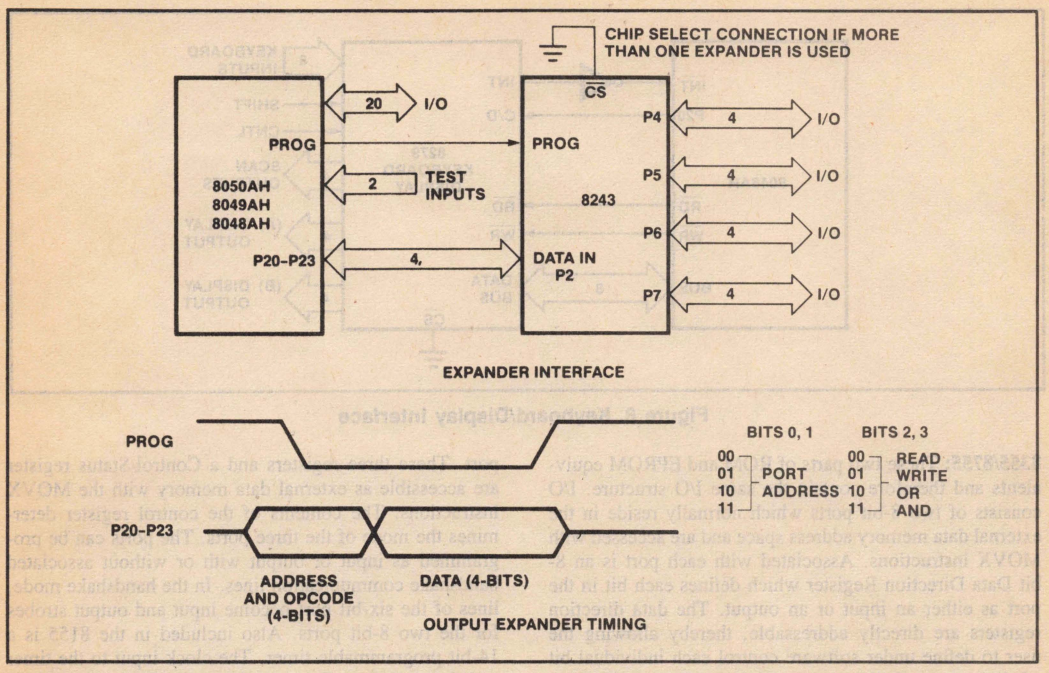
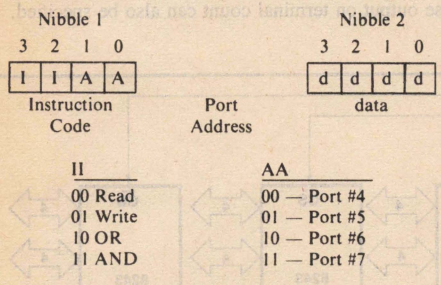


Figure 7. 8243 Expander I/O Interface



A high to low transition of the PROG line indicates that address is present, while allow to high transition indicates the presence of data. Additional 8243's may be added to the four-bit bus and chip selected using additional output lines from the 8048AH/8748H.

I/O PORT CHARACTERISTICS

Each of the four 4-bit ports of the 8243 can serve as either input or output and can provide high drive capability in both the high and low state.

4.2 I/O Expansion with Standard Peripherals

Standard MCS-80/85 type I/O devices may be added to the MCS®-48 using the same bus and timing used for Data Memory expansion. Figure 8 shows an example of how an 8048AH can be connected to an MCS-85 peripheral. I/O devices reside on the Data Memory bus and in the data memory address space and are accessed with the same MOVX instructions. (See the previous section on data memory expansion for a description of timing.) The following are a few of the Standard MCS-80 devices which are very useful in MCS®-48 systems:

- 8214 Priority Interrupt Encoder
- 8251 Serial Communications Interface
- 8255 General Purpose Programmable I/O
- 8279 Keyboard/Display Interface
- 8254 Interval Timer

4.3 Combination Memory and I/O Expanders

As mentioned in the sections on program and data memory expansion, the 8355/8755 and 8155 expanders also contain I/O capability.

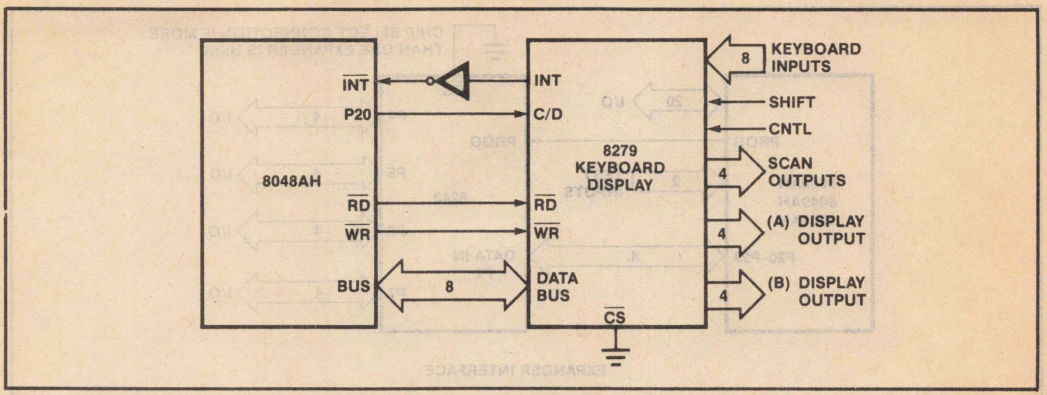


Figure 8. Keyboard/Display Interface

8355/8755: These two parts of ROM and EPROM equivalents and therefore contain the same I/O structure. I/O consists of two 8-bit ports which normally reside in the external data memory address space and are accessed with MOVX instructions. Associated with each port is an 8-bit Data Direction Register which defines each bit in the port as either an input or an output. The data direction registers are directly addressable, thereby allowing the user to define under software control each individual bit of the ports as either input or output. All outputs are statically latched and double buffered. Inputs are not latched.

8155/8156: I/O on the 8155/8156 is configured as two 8-bit programmable I/O ports and one 6-bit programmable port. These three registers and a Control/Status register are accessible as external data memory with the MOVX instructions. The contents of the control register determines the mode of the three ports. The ports can be programmed as input or output with or without associated handshake communication lines. In the handshake mode, lines of the six-bit port become input and output strobes for the two 8-bit ports. Also included in the 8155 is a 14-bit programmable timer. The clock input to the timer and the timer overflow output are available on external pins. The timer can be programmed to stop on terminal count or to continuously reload itself. A square wave or pulse output on terminal count can also be specified.

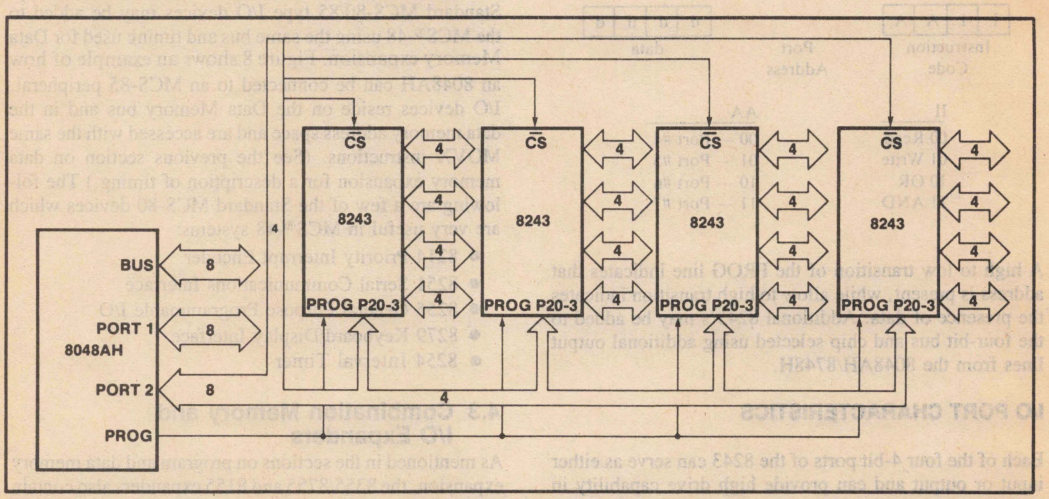


Figure 9. Low Cost I/O Expansion

I/O EXPANSION EXAMPLES

Figure 9 shows the expansion of I/O using multiple 8243's. The only difference from a single 8243 system is the addition of chip selects provided by additional 8048AH output lines. Two output liens and a decoder could also be used to address the four chips. Large numbers of 8243's would require a chip select decoder chip such as the 8205 to save I/O pins.

Figure 10 shows the 8048AH interface to a standard MCS®-80 peripheral; in this case, the 8255 Programmable Peripheral Interface, a 40-pin part which provides three 8-bit programmable I/O ports. The 8255 bus interface is typical of programmable MCS®-80 peripherals with an 8-bit bidirectional data bus, a RD and WR input for Read/Write control, a CS (chip select) input used to enable the Read/Write control logic and the address inputs used to select various internal registers.

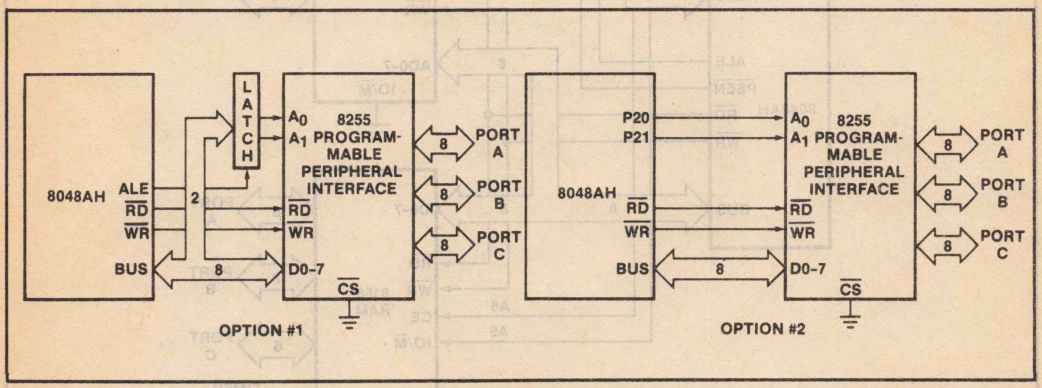


Figure 10. Interface to MCS®-80 Peripherals

Interconnection to the 8048AH is very straightforward with BUS, $\overline{RD}$, and $\overline{WR}$ connecting directly to the corresponding pins on the 8255. The only design consideration is the way in which the internal registers of the 8255 are to be addressed. If the registers are to be addressed as external data memory using the MOVX instructions, the appropriate number of address bits (in this case, 2) must be latched on BUS using ALE as described in the section on external data memories. If only a single device is connected to BUS, the 8255 may be continuously selected by grounding $\overline{CS}$. If multiple 8255's are used, additional address bits can be latched and used as chip selects.

A second addressing method eliminates external latches and chip select decoders by using output port lines as address and chip select lines directly. This method, of course, requires the setting of an output port with address information prior to executing a MOVX instruction.

5.0 MULTI-CHIP MCS®-48 SYSTEMS

Figure 11 shows the addition of two memory expanders to the 8048AH, one 8355/8755 ROM and one 8156 RAM. The main consideration in designing such a system is the addressing of the various memories and I/O ports. Note that in this configuration address lines A_{10} and A_{11} have been ORed to chip select the 8355. This ensures that the chip is active for all external program memory fetches in the 1K to 3K range and is disabled for all other addresses. This gating has been added to allow the I/O port of the 8355 to be used. If the chip was left selected all the time, there would be conflict between these ports and the RAM and I/O of the 8156. The NOR gate could be eliminated and A_{11} connected directly to the CE (instead of $\overline{CE}$) input of the 8355; however, this would create a 1K word "hole" in the program memory by causing the 8355 to be active in the 2K and 4K range instead of the normal 1K to 3K range.

In this system the various locations are addressed as follows:

- Data RAM — Addresses 0 to 255 when Port 2 Bit 0 has been previously set = 1 and Bit 1 set = 0
- RAM I/O — Addresses 0 to 3 when Port 2 Bit 0 = 1 and Bit 1 = 1
- ROM I/O — Addresses 0 to 3 when Port 2 Bit 2 or Bit 3 = 1

See the memory map in Figure 12.

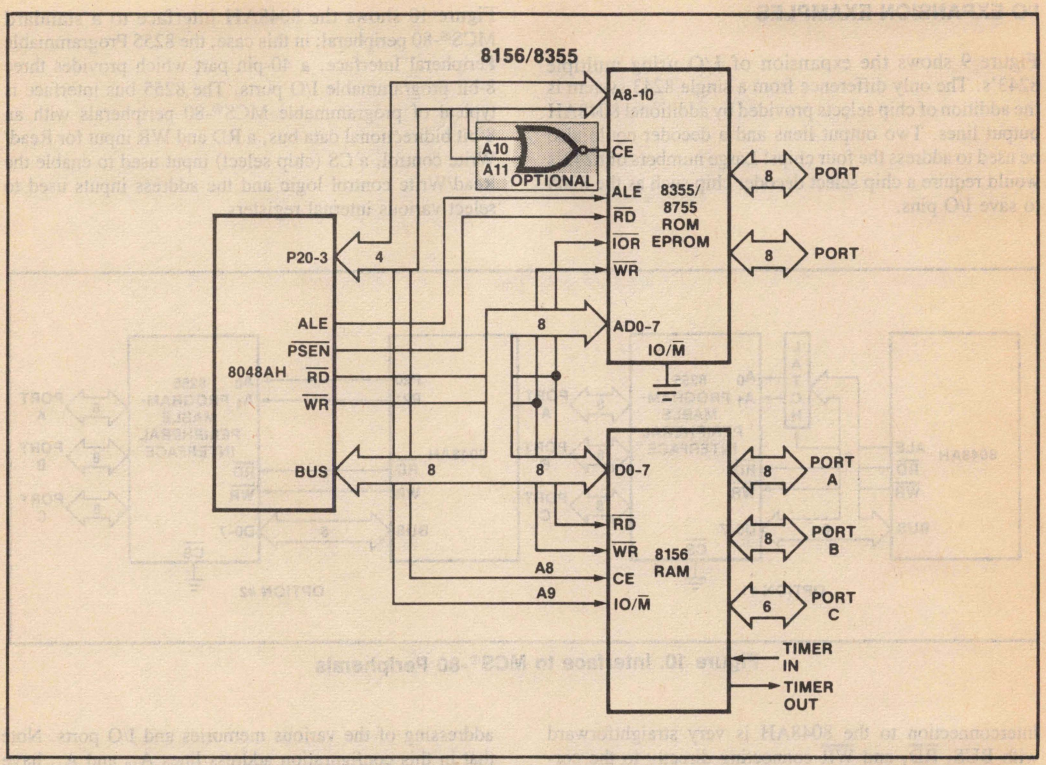


Figure 11. The Three-Component MCS®-48 System

6.0 MEMORY BANK SWITCHING

Certain systems may require more than the 4K words of program memory which are directly addressable by the program counter or more than the 256 data memory and I/O locations directly addressable by the pointer registers R0 and R1. These systems can be achieved using "bank switching" techniques. Bank switching is merely the selection of various blocks of "banks" of memory using dedicated output port lines from the processor. In the case of the 8048AH, program memory is selected in blocks of 4K words at a time, while data memory and I/O are enabled 256 words at a time.

The most important consideration in implementing two or more banks is the software required to cross the bank boundaries. Each crossing of the boundary requires that the processor first write a control bit to an output port before accessing memory or I/O in the new bank. If program memory is being switched, programs should be organized to keep boundary crossings to a minimum.

Jumping to subroutines across the boundary should be avoided when possible since the programmer must keep track of which bank to return to after completion of the subroutine. If these subroutines are to be nested and accessed from either bank, a software "stack" should be implemented to save the bank switch bit just as if it were another bit of the program counter.

From a hardware standpoint bank switching is very straightforward and involves only the connection of an I/O line or lines as bank enable signals. These enables are ANDed with normal memory and I/O chip select signals to activate the proper bank.

7.0 CONTROL SIGNAL SUMMARY

Table 1 summarizes the instructions which activate the various control outputs of the MCS®-48 processors. During all other instructions these outputs are driven to the active state.

Table 1. MCS®-48 Control Signals

Control Signal	When Active
$\overline{RD}$	During MOVX, A, @R or INS Bus
$\overline{WR}$	During MOVX @R, A or OUTL Bus
ALE	Every Machine Cycle
$\overline{PSEN}$	During Fetch of external program memory (instruction or immediate data)
PROG	During MOVD, A,P ANLD P,A MOVD P,A ORLD P,A

8.0 PORT CHARACTERISTICS

8.1 BUS Port Operations

The BUS port can operate in three different modes: as a latched I/O port, as a bidirectional bus port, or as a program memory address output when external memory is used. The BUS port lines are either active high, active low, or high impedance (floating).

The latched mode (INS, OUTL) is intended for use in the single-chip configuration where BUS is not begin used as an expander port. OUTL and MOVX instructions can be mixed if necessary. However, a previously latched output will be destroyed by executing a MOVX instruction and BUS will be left in the high impedance state. INS does not put the BUS in a high impedance state. Therefore, the use of MOVX after OUTL to put the BUS in a high impedance state is necessary before an INS instruction intended to read an external word (as opposed to the previously latched value).

OUTL should never be used in a system with external program memory, since latching BUS can cause the next instruction, if external, to be fetched improperly.

8.2 Port 2 Operations

The lower half of Port 2 can be used in three different ways: as a quasi-bidirectional static port, as an 8243 expander port, and to adddress external program memory.

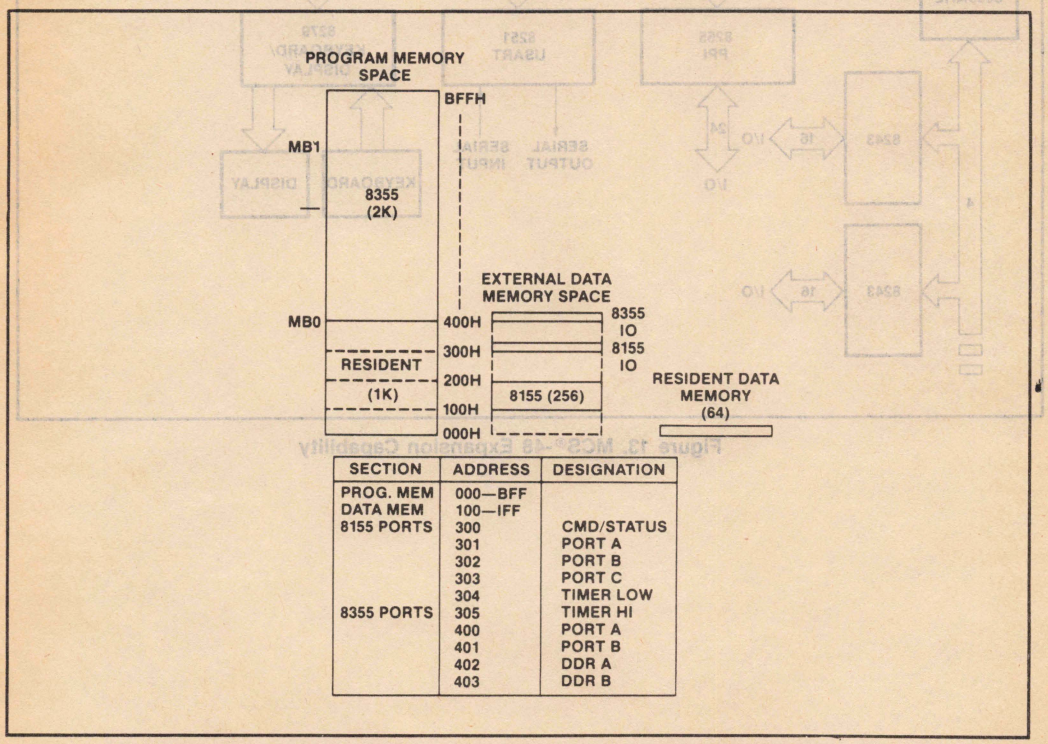


Figure 12. Memory Map for Three-Component MCS®-48 Family

In all cases outputs are driven low by an active device and driven high momentarily by a low impedance device and held high by a high impedance device to VCC.

The port may contain latched I/O data prior to its use in another mode without affecting operation of either. If lower Port 2 (P20-3) is used to output address for an external program memory fetch. the I/O information previously latched will be automatically removed temporarily while address is present, then retored when the fetch is complete. However, if lower Port 2 is used to communicate with an 8243, previously latched I/O information will be removed and not restored. After an input from the 8243, P20-3 will be left in the input mode (floating). After an output to the 8243, P20-3 will contain the value written, ANDed, or ORed to the 8243 port.

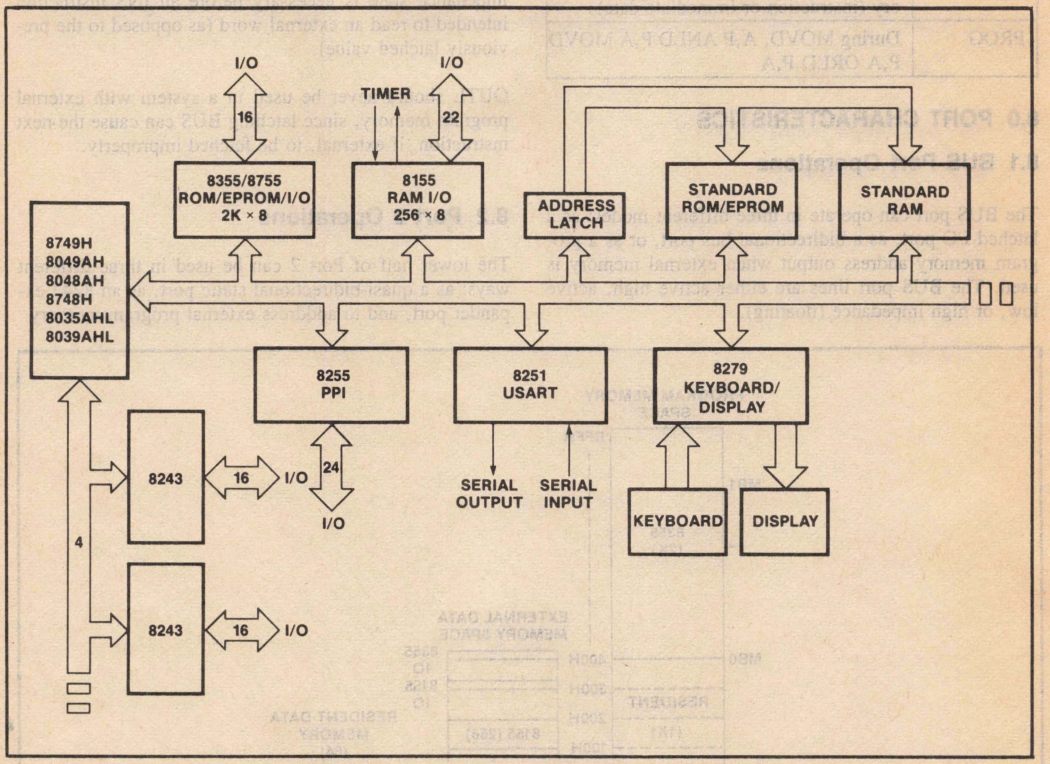


Figure 13. MCS®-48 Expansion Capability

MCS®-48 Instruction Set

3

MCS®-48 INSTRUCTION SET

1.0 INTRODUCTION

The MCS®-48 instruction set is extensive for a machine of its size and has been tailored to be straightforward and very efficient in its use of program memory. All instructions are either one or two bytes in length and over 80% are only one byte long. Also, all instructions execute in either one or two cycles and over 50% of all instructions execute in a single cycle. Double cycle instructions include all immediate instructions, and all I/O instructions.

The MCS-48 microcomputers have been designed to handle arithmetic operations efficiently in both binary and BCD as well as handle the single-bit operations required in control applications. Special instructions have also been included to simplify loop counters, table look-up routines, and N-way branch routines.

1.1 Data Transfers

As can be seen in Figure 1 the 8-bit accumulator is the central point for all data transfers within the 8048. Data can be transferred between the 8 registers of each working register bank and the accumulator directly, i.e., the source or destination register is specified by the instruction. The remaining locations of the internal RAM array are referred to as Data Memory and are addressed indirectly via an address stored in either R0 or R1 of the active register bank. R0 and R1 are also used to indirecly address external data memory when it is present. Transfers to and from internal RAM require one cycle, while transfers to external RAM require two. Constants stored in Program Memory can be loaded directly to the accumulator and to the 8 working registers. Data can also be transferred directly between the accumulator and the on-

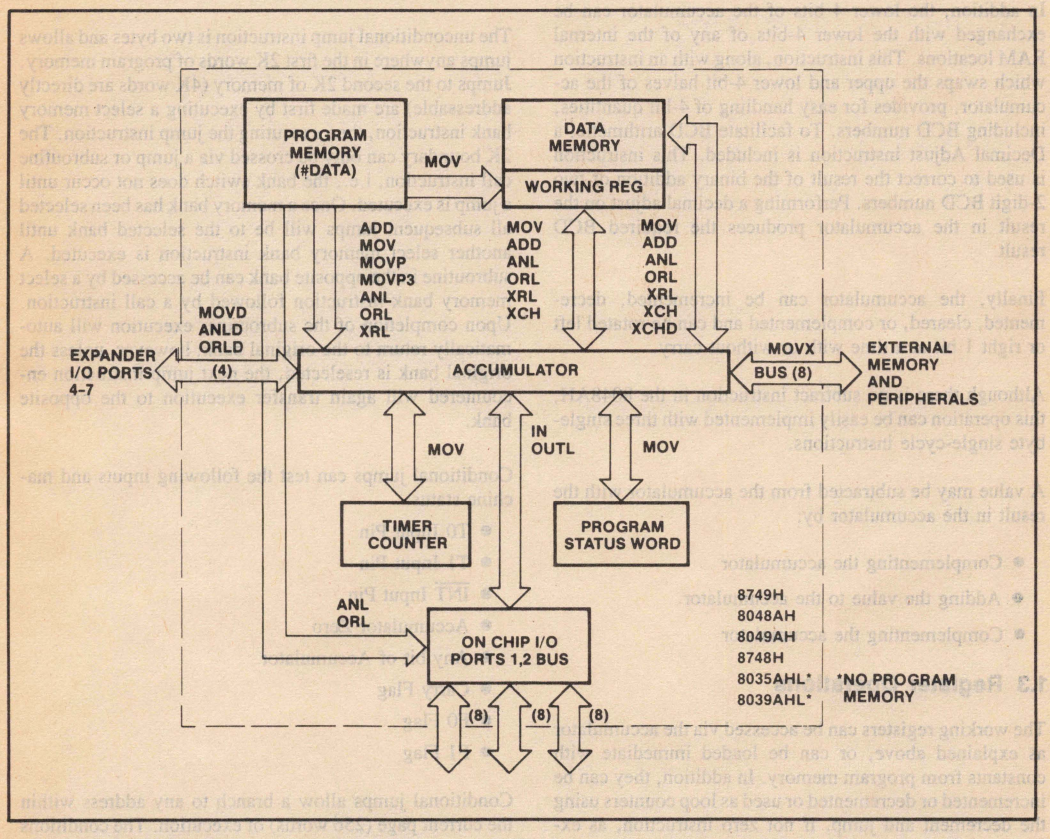


Figure 1. Data Transfer Instructions

board timer counter or the accumulator and the Program Status word (PSW). Writing to the PSW alters machine status accordingly and provides a means of restoring status after an interrupt or of altering the stack pointer if necessary.

1.2 Accumulator Operations

Immediate data, data memory, or the working registers can be added with or without carry to the accumulator. These sources can also be ANDed, ORed, or Exclusive ORed to the accumulator. Data may be moved to or from the accumulator and working registers or data memory. The two values can also be exchanged in a single operation.

In addition, the lower 4 bits of the accumulator can be exchanged with the lower 4-bits of any of the internal RAM locations. This instruction, along with an instruction which swaps the upper and lower 4-bit halves of the accumulator, provides for easy handling of 4-bit quantities, including BCD numbers. To facilitate BCD arithmetic, a Decimal Adjust instruction is included. This instruction is used to correct the result of the binary addition of two 2-digit BCD numbers. Performing a decimal adjust on the result in the accumulator produces the required BCD result.

Finally, the accumulator can be incremented, decremented, cleared, or complemented and can be rotated left or right 1 bit at a time with or without carry.

Although there is no subtract instruction in the 8048AH, this operation can be easily implemented with three single-byte single-cycle instructions.

A value may be subtracted from the accumulator with the result in the accumulator by:

- Complementing the accumulator
- Adding the value to the accumulator
- Complementing the accumulator

1.3 Register Operations

The working registers can be accessed via the accumulator as explained above, or can be loaded immediate with constants from program memory. In addition, they can be incremented or decremented or used as loop counters using the decrement and jump, if not zero instruction, as explained under branch instructions.

All Data Memory including working registers can be accessed with indirect instructions via R0 and R1 and can be incremented.

1.4 Flags

There are four user-accessible flags in the 8048AH: Carry, Auxiliary Carry, F0 and F1. Carry indicates overflow of the accumulator, and Auxiliary Carry is used to indiate overflow between BCD digits and is used during decimal-adjust operation. Both Carry and Auxiliary Carry are accessible as part of the program status word and are stored on the stack during subroutines. F0 and F1 are undedicated general-purpose flags to be used as the programmer desires. Both flags can be cleared or complemented and tested by conditional jump instructions. F0 is also accessible via the Program Status word and is stored on the stack with the carry flags.

1.5 Branch Instructions

The unconditional jump instruction is two bytes and allows jumps anywhere in the first 2K words of program memory. Jumps to the second 2K of memory (4K words are directly addressable) are made first by executing a select memory bank instruction, then executing the jump instruction. The 2K boundary can only be crossed via a jump or subroutine call instruction, i.e., the bank switch does not occur until a jump is executed. Once a memory bank has been selected all subsequent jumps will be to the selected bank until another select memory bank instruction is executed. A subroutine in the opposite bank can be accessed by a select memory bank instruction followed by a call instruction. Upon completion of the subroutine, execution will automatically return to the original bank; however, unless the original bank is reselected, the next jump instruction encountered will again transfer execution to the opposite bank.

Conditional jumps can test the following inputs and machine status:

- T0 Input Pin
- T1 Input Pin
- $\overline{\text{INT}}$ Input Pin
- Accumulator Zero
- Any bit of Accumulator
- Carry Flag
- F0 Flag
- F1 Flag

Conditional jumps allow a branch to any address within the current page (256 words) of execution. The conditions tested are the instantaneous values at the time the conditional jump is executed. For instance, the jump on accumulator zero instruction tests the accumulator itself, not an intermediate zero flag.

The decrement register and jump if not zero instruction combines a decrement and a branch instruction to create an instruction very useful in implementing a loop counter. This instruction can designate any one of the 8 working registers as a counter and can effect a branch to any address within the current page of execution.

A single-byte indirect jump instruction allows the program to be vectored to any one of several different locations based on the contents of the accumulator. The contents of the accumulator points to a location in program memory which contains the jump address. The 8-bit jump address refers to the current page of execution. This instruction could be used, for instance, to vector to any one of several routines based on an ASCII character which has been loaded in the accumulator. In this way ASCII key inputs can be used to initiate various routines.

1.6 Subroutines

Subroutines are entered by executing a call instruction. Calls can be made like unconditional jumps to any address in a 2K word bank, and jumps across the 2K boundary are executed in the same manner. Two separate return instructions determine whether or not status (upper 4-bits of PSW) is restored upon return from the subroutine.

The return and restore status instruction also signals the end of an interrupt service routine if one has been in progress.

1.7 Timer Instructions

The 8-bit on board timer/counter can be loaded or read via the accumulator while the counter is stopped or while counting. The counter can be started as a timer with an internal clock source or an event counter or timer with an external clock applied to the T1 input pin. The instruction executed determines which clock source is used. A single instruction stops the counter whether it is operating with an internal or an external clock source. In addition, two instructions allow the timer interrupt to be enabled or disabled.

1.8 Control Instructions

Two instructions allow the external interrupt source to be enabled or disabled. Interrupts are initially disabled and are automatically disabled while an interrupt service routines is in progress and re-enabled afterward.

There are four memory bank select instructions, two to designate the active working register bank and two to control program memory banks. The operation of the program memory bank switch is explained in Section 2.2 in the Expanded MCS-48 System chapter.

The working register bank switch instructions allow the programmer to immediately substitute a second 8-register working register bank for the one in use. This effectively provides 16 working registers or it can be used as a means of quickly saving the contents of the registers in response to an interrupt. The user has the option to switch or not to switch banks on interrupt. However, if the banks are switched, the original bank will be automatically restored upon execution of a return and restore status instruction at the end of the interrupt service routine.

A special instruction enables an internal clock, which is the XTAL frequency divided by three to be output on pin T0. This clock can be used as a general-purpose clock in the user's system. This instruction should be used only to initialize the system since the clock output can be disabled only by application of system reset.

1.9 Input/Output Instructions

Ports 1 and 2 are 8-bit static I/O ports which can be loaded to and from the accumulator. Outputs are statically latched but inputs are not latched and must be read while inputs are present. In addition, immediate data from program memory can be ANDed or ORed directly to Port 1 and Port 2 with the result remaining on the port. This allows "masks" stored in program memory to selectively set or reset individual bits of the I/O ports. Ports 1 and 2 are configured to allow input on a given pin by first writing a "1" out to the pin.

An 8-bit port called BUS can also be accessed via the accumulator and can have statically latched outputs as well. It too can have immediate data ANDed or ORed directly to its outputs, however, unlike ports 1 and 2, all eight lines of BUS must be treated as either input or output at any one time. In addition to being a static port, BUS can be used as a true synchronous bi-directional port using the Move External instructions used to access external data memory. When these instructions are executed, a corresponding $\overline{\text{READ}}$ or $\overline{\text{WRITE}}$ pulse is generated and data is valid only at that time. When data is not being transferred, BUS is in a high impedance state. Note that the OUTL, ANL, and the ORL instructions for the BUS are for use with internal program memory only.

The basic three on-board I/O ports can be expanded via a 4-bit expander bus using half of port 2. I/O expander devices on this bus consist of four 4-bit ports which are addressed as ports 4 through 7. These ports have their own AND and OR instructions like the on-board ports as well as move instructions to transfer data in or out. The expander AND and OR instructions, however, combine the contents of accumulator with the selected port rather than immediate data as is done with the on-board ports.

I/O devices can also be added externally using the BUS port as the expansion bus. In this case the I/O ports become "memory mapped", i.e., they are addressed in the same way as external data memory and exist in the external data memory address space addressed by pointer register R0 or R1.

2.0 INSTRUCTION SET DESCRIPTION

The following pages describe the MCS®-48 instruction set in detail. The instruction set is first summarized with instructions grouped functionally. This summary page is followed by a detailed description listed alphabetically by mnemonic opcode.

The alphabetical listing includes the following information.

- Mnemonic
- Machine Code
- Verbal Description
- Symbolic Description
- Assembly Language Example

The machine code is represented with the most significant bit (7) to the left and two byte instructions are represented with the first byte on the left. The assembly language examples are formulated as follows:

Arbitrary

Label: Mnemonic, Operand;

Descriptive Comment

8048AH/8748H/8049AH/8050AH/8749H
Instruction Set Summary

Mnemonic	Description	Bytes	Cycle
Accumulator			
ADD A, R	Add register to A	1	1
ADD A, @R	Add data memory to A	1	1
ADD A, # data	Add immediate to A	2	2
ADDC A, R	Add register with carry	1	1
ADDC A, @R	Add data memory with carry	1	1
ADDC A, # data	Add immediate with carry	2	2
ANL A, R	And register to A	1	1
ANL A, @R	And data memory to A	1	1
ANL A, # data	And immediate to A	2	2
ORL A, R	Or register to A	1	1
ORL A @R	Or data memory to A	1	1
ORL A, # data	Or immediate to A	2	2
XRL A, R	Exclusive Or register to A	1	1
XRL A, @R	Exclusive or data memory to A	1	1
XRL, A, # data	Exclusive or immediate to A	2	2
INC A	Increment A	1	1
DEC A	Decrement A	1	1
CLR A	Clear A	1	1
CPL A	Complement A	1	1
DA A	Decimal adjust A	1	1
SWAP A	Swap nibbles of A	1	1
RL A	Rotate A left	1	1
RLC A	Rotate A left through carry	1	1
RR A	Rotate A right	1	1
RRC A	Rotate A right through carry	1	1
Input/Output			
IN A, P	Input port to A	1	2
OUTL P, A	Output A to port	1	2
ANL P, # data	And immediate to port	2	2
ORL P, # data	Or immediate to port	2	2
*INS A, BUS	Input BUS to A	1	2
*OUTL BUS, A	Output A to BUS	1	2
*ANL BUS, # data	And immediate to BUS	2	2
*ORL BUS, # data	Or immediate to BUS	2	2
MOVD A, P	Input Expander port to A	1	2
MOVD P, A	Output A to Expander port	1	2
ANLD P, A	And A to Expander port	1	2
ORLD P, A	Or A to Expander port	1	2

Mnemonic	Description	Bytes	Cycles
Registers			
INC R	Increment register	1	1
INC @R	Increment data memory	1	1
DEC R	Decrement register	1	1
Branch			
JMP addr	Jump unconditional	2	2
JMPP @A	Jump indirect	1	2
DJNZ R, addr	Decrement register and jump	2	2
JC addr	Jump on carry = 1	2	2
JNC addr	Jump on carry = 0	2	2
JZ addr	Jump on A Zero	2	2
JNZ addr	Jump on A not Zero	2	2
JT0 addr	Jump on T0 = 1	2	2
JNT0 addr	Jump on T0 = 0	2	2
JT1 addr	Jump on T1 = 1	2	2
JNT1 addr	Jump on T1 = 0	2	2
JF0 addr	Jump on F0 = 1	2	2
JF1 addr	Jump on F1 = 1	2	2
JTF addr	Jump on timer flag = 1	2	2
JNI addr	Jump on INT = 0	2	2
JBb addr	Jump on Accumulator Bit	2	2
Subroutine			
CALL addr	Jump to subroutine	2	2
RET	Return	1	2
RETR	Return and restore status	1	2
Flags			
CLR C	Clear Carry	1	1
CPL C	Complement Carry	1	1
CLR F0	Clear Flag 0	1	1
CPL F0	Complement Flag 0	1	1
CLR F1	Clear Flag 1	1	1
CPL F1	Complement Flag 1	1	1
Data Moves			
MOV A, R	Move register to A	1	1
MOV A, @R	Move data memory to A	1	1
MOV A, # data	Move immediate to A	2	2
MOV R, A	Move A to register	1	1
MOV @R, A	Move A to data memory	1	1
MOV R, # data	Move immediate to register	2	2
MOV @R, # data	Move immediate to data memory	2	2
MOV A, PSW	Move PSW to A	1	1
MOV PSW, A	Move A to PSW	1	1

Mnemonics copyright Intel Corporation 1983.

*For use with internal memory only.

8048AH/8748H/8049AH/8050AH/8749H
Instruction Set Summary (Con't)

Mnemonic	Description	Bytes	Cycle
Data Moves (Cont'd)			
XCH A, R	Exchange A and register	1	1
XCH A, @R	Exchange A and data memory	1	1
XCHD A, @R	Exchange nibble of A and register	1	1
MOVX A, @R	Move external data memory to A	1	2
MOVX @R, A	Move A to external data memory	1	2
MOVP A, @A	Move to A from current page	1	2
MOVP3 A, @A	Move to A from Page 3	1	2
Timer/Counter			
MOV A, T	Read Timer/Counter	1	1
MOV T, A	Load Timer/Counter	1	1
STRT T	Start Timer	1	1
STRT CNT	Start Counter	1	1
STOP TCNT	Stop Timer/Counter	1	1
EN TCNTI	Enable Timer/Counter Interrupt	1	1
DIS TCNTI	Disable Timer/Counter Interrupt	1	1

Mnemonic	Description	Bytes	Cycle
Control			
EN I	Enable external Interrupt	1	1
DIS I	Disable external Interrupt	1	1
SEL RB0	Select register bank 0	1	1
SEL RB1	Select register bank 1	1	1
SEL MB0	Select memory bank 0	1	1
SEL MB1	Select memory bank 1	1	1
ENT0 CLK	Enable clock output on T0	1	1
NOP	No Operation	1	1

Mnemonics copyright Intel Corporation 1983.

MCS®-48 INSTRUCTION SET
Symbols and Abbreviations Used

A	Accumulator
AC	Auxiliary Carry
addr	12-Bit Program Memory Address
Bb	Bit Designator (b = 0–7)
BS	Bank Switch
BUS	BUS Port
C	Carry
CLK	Clock
CNT	Event Counter
CRR	Conversion Result Register
D	Mnemonic for 4-Bit Digit (Nibble)
data	8-Bit Number or Expression
DBF	Memory Bank Flip-Flop
F0, F1	Flag 0, Flag 1
I	Interrupt
P	Mnemonic for "in-page" Operation
PC	Program Counter
Pp	Port Designator (p = 1, 2 or 4–7)
PSW	Program Status Word
Ri	Data memory Pointer (i = 0, or 1)
Rr	Register Designator (r = 0–7)
SP	Stack Pointer
T	Timer
TF	Timer Flag
T0, T1	Test 0, Test 1
X	Mnemonic for External RAM
#	Immediate Data Prefix
@	Indirect Address Prefix
$	Current Value of Program Counter
(X)	Contents of X
((X))	Contents of Location Addressed by X
←	Is Replaced by

Mnemonics copyright Intel Corporation 1983.

ADD A,R_r Add Register Contents to Accumulator

Encoding: | 0 1 1 0 | 1 r r r | 68H-6FH

Description: The contents of register 'r' are added to the accumulator. Carry is affected.

Operation: (A) ← (A) + (Rr) r = 0-7

Example:
```
ADDREG: ADD A,R6      ;ADD REG 6 CONTENTS
                      ;TO ACC
```

ADD A,@R_i Add Data Memory Contents to Accumulator

Encoding; | 0 1 1 0 | 0 0 0 i | 60H-61H

Description: The contents of the resident data memory location addressed by register 'i' bits 0–5** are added to the accumulator. Carry is affected.

Operation: (A) ← (A) + ((Ri)) i = 0-1

Example:
```
ADDM: MOV R0, #01FH   ;MOVE '1F' HEX TO REG 0
      ADD A, @R0      ;ADD VALUE OF LOCATION
                      ;31 TO ACC
```

ADD A,#data Add Immediate Data to Accumulator

Encoding: | 0 0 0 0 | 0 0 1 1 | d_7 d_6 d_5 d_4 | d_3 d_2 d_1 d_0 | 03H

Description: This is a 2-cycle instruction. The specified data is added to the accumulator. Carry is affected.

Operation: (A) ← (A) + data

Example:
```
ADDID: ADD A,#ADDER:  ;ADD VALUE OF SYMBOL
                      ;ADDER' TO ACC
```

ADDC A,R_r Add Carry and Register Contents to Accumulator

Encoding: | 0 1 1 1 | 1 r r r | 78H-7FH

Description: The content of the carry bit is added to accumulator location 0 and the carry bit cleared. The contents of register 'r' are then added to the accumulator. Carry is affected.

Operation: (A) ← (A) + (Rr) + (C) r = 0-7

Example:
```
ADDRGC: ADDC A,R4     ;ADD CARRY AND REG 4
                      ;CONTENTS TO ACC
```

** 0–5 in 8048AH/8748H
0–6 in 8049AH/8749H
0–7 in 8050AH

ADDC A,@R_i Add Carry and Data Memory Contents to Accumulator

Encoding: | 0 1 1 1 | 0 0 0 i | 70H-71H

Description: The content of the carry bit is added to accumulator location 0 and the carry bit cleared. Then the contents of the resident data memory location addressed by register 'i' bits 0–5** are added to the accumulator. Carry is affected.

Operation: (A) ← (A) + ((Ri)) + (C) i = 0-1

Example:
```
ADDMC:  MOV R1,#40      ;MOVE '40' DEC TO REG 1
        ADDC A,@R1      ;ADD CARRY AND LOCATION 40
                        ;CONTENTS TO ACC
```

ADDC A,@data Add Carry and Immediate Data to Accumulator

Encoding: | 0 0 0 1 | 0 0 1 1 | | d_7 d_6 d_5 d_4 | d_3 d_2 d_1 d_0 | 13H

Description: This is a 2-cycle instruction. The content of the carry bit is added to accumulator location 0 and the carry bit cleared. Then the specified data is added to the accumulator. Carry is affected.

Operation: (A) ← (A) + data + (C)

Example:
```
ADDC A,#225     ;ADD CARRY AND '225' DEC
                ;TO ACC
```

ANL A,R_r Logical AND Accumulator with Register Mask

Encoding: | 0 1 0 1 | 1 r r r | 58H-5FH

Description: Data in the accumulator is logically ANDed with the mask contained in working register 'r'.

Operation: (A) ← (A) AND (Rr) r = 0-7

Example:
```
ANDREG: ANL A,R3        ;'AND' ACC CONTENTS WITH MASK
                        ;IN REG 3
```

ANL A,@R_i Logical AND Accumulator with memory Mask

Encoding: | 0 1 0 1 | 0 0 0 i | 50H-51H

Description: Data in the accumulator is logically ANDed with the mask contained in the data memory location referenced by register 'i' bits 0-5**.

Operation: (A) ← (A) AND ((Ri)) i = 0-1

Example:
```
ANDDM:  MOV R0,#03FH    ;MOVE '3F' HEX TO REG 0
        ANL A, @R0      ;'AND' ACC CONTENTS WITH
                        ;MASK IN LOCATION 63
```

** 0–5 in 8048AH/8748H
0–6 in 8049AH/8749H
0–7 in 8050AH

ANL A,#data Logical AND Accumulator with Immediate Mask

Encoding: | 0 1 0 1 | 0 0 1 1 | | d_7 d_6 d_5 d_4 | d_3 d_2 d_1 d_0 | 53H

Description: This is a 2-cycle instruction. Data in the accumulator is logically ANDed with an immediately-specified mask.

Operation: (A) ← (A) AND data

Examples:
```
ANDID: ANL A,#0AFH      ;'AND' ACC CONTENTS
                        ;WITH MASK 10101111
       ANL A,#3 + X/Y   ;'AND' ACC CONTENTS
                        ;WITH VALUE OF EXP
                        ;'3 + XY/Y'
```

ANL BUS,#data* Logical AND BUS with Immediate Mask

Encoding: | 1 0 0 1 | 1 0 0 0 | | d_7 d_6 d_5 d_4 | d_3 d_2 d_1 d_0 | 98H

Description: This is a 2-cycle instruction. Data on the BUS port is logically ANDed with an immediately-specified mask. This instruction assumes prior specification of an 'OUTL BUS, A' instruction.

Operation: (BUS) ← (BUS) AND data

Example:
```
ANDBUS: ANL BUS,#MASK   ;'AND' BUS CONTENTS
                        ;WITH MASK EQUAL VALUE
                        ;OF SYMBOL 'MASK'
```

ANL Pp,#data Logical AND Port 1-2 with Immediate Mask

Encoding: | 1 0 0 1 | 1 0 p p | | d_7 d_6 d_5 d_4 | d_3 d_2 d_1 d_0 | 99H-9AH

Description: This is a 2-cycle instruction. Data on port 'p' is logically ANDed with an immediately-specified mask.

Operation: (Pp) ← (Pp) AND DATA p = 1-2

Example:
```
ANDP2: ANL P2,#0F0H     ;'AND' PORT 2 CONTENTS
                        ;WITH MASK 'F0' HEX
                        ;(CLEAR P20-23)
```

* For use with internal program memory ONLY.

ANLD Pp,A Logical AND Port 4-7 with Accumulator Mask

Encoding: | 1 0 0 1 | 1 1 p p | 9CH-9FH

Description: This is a 2-cycle instruction. Data on port 'p' is logically ANDed with the digit mask contained in accumulator bits 0-3.

Operation: (Pp) ← (Pp) AND (A0-3) p = 4-7

Note: The mapping of port 'p' to opcode bits 0-1 is as follows:

1 0	Port
0 0	4
0 1	5
1 0	6
1 1	7

Example:
```
ANDP4: ANLD P4,A      ;'AND' PORT 4 CONTENTS
                      ;WITH ACC BITS 0-3
```

CALL address Subroutine Call

Encoding: | a_{10} a_9 a_8 1 | 0 1 0 0 | | a_7 a_6 a_5 a_4 | a_3 a_2 a_1 a_0 |

Page	Hex Op Code
0	14
1	34
2	54
3	74
4	94
5	B4
6	D4
7	F4

Description: This is a 2-cycle instruction. The program counter and PSW bits 4-7 are saved in the stack. The stack pointer (PSW bits 0-2) is updated. Program control is then passed to the location specified by 'address'. PC bit 11 is determined by the most recent SEL MB instruction.

A CALL cannot begin in locations 2046-2047 or 4094-4095. Execution continues at the instruction following the CALL upon return from the subroutine.

Operation:
$((SP)) \leftarrow (PC), (PSW_{4-7})$
$(SP) \leftarrow (SP) + 1$
$(PC_{8-10}) \leftarrow (addr_{8-10})$
$(PC_{0-7}) \leftarrow (addr_{0-7})$
$(PC_{11}) \leftarrow DBF$

Example: Add three groups of two numbers. Put subtotals in locations 50, 51 and total in location 52.

```
          MOV R0,#50        ;MOVE '50' DEC TO ADDRESS
                            ;REG 0
BEGADD:   MOV A,R1          ;MOVE CONTENTS OF REG 1
                            ;TO ACC
          ADD A,R2          ;ADD REG 2 TO ACC
          CALL SUBTOT       ;CALL SUBROUTINE 'SUBTOT'
          ADDC A R3         ;ADD REG 3 TO ACC
          ADDC A,R4         ;ADD REG 4 TO ACC
          CALL SUBTOT       ;CALL SUBROUTINE 'SUBTOT'
          ADDC A,R5         ;ADD REG 5 TO ACC
          ADDC A,R6         ;ADD REG 6 TO ACC
          CALL SUBTOT       ;CALL SUBROUTINE 'SUBTOT'
SUBTOT:   MOV @R0,A         ;MOVE CONTENTS OF ACC TO
                            ;LOCATION ADDRESSED BY
                            ;REG 0
          INC R0            ;INCREMENT REG 0
          RET               ;RETURN TO MAIN PROGRAM
```

CLR A Clear Accumulator

Encoding: | 0 0 1 0 | 0 1 1 1 | 27H

Description: The contents of the accumulator are cleared to zero.

Operation: A ← 0

CLR C Clear Carry Bit

Encoding: | 1 0 0 1 | 0 1 1 1 | 97H

Description: During normal program execution, the carry bit can be set to one by the ADD, ADDC, RLC, CPL C, RRC, and DAA insructions. This instruction resets the carry bit to zero.

Operation: C ← 0

CLR F1 Clear Flag 1

Encoding: | 1 0 1 0 | 0 1 0 1 | A5H

Description: Flag 1 is cleared to zero.

Operation: (F1) ← 0

CLR F0 Clear Flag 0

Encoding: | 1 0 0 0 | 0 1 0 1 | 85H

Description: Flag 0 is cleared to zero.

Operation: (F0) ← 0

CPL A Complement Accumulator

Encoding: | 0 0 1 1 | 0 1 1 1 | 37H

Description: The contents of the accumulator are complemented. This is strictly a one's complement. Each one is changed to zero and vice-versa.

Operation: (A) ← NOT (A)

Example: Assume accumulator contains 01101010.

```
CPLA: CPL A          ;ACC CONTENTS ARE COMPLE-
                     ;MENTED TO 10010101
```

CPL C Complement Carry Bit

Encoding: | 1 0 1 0 | 0 1 1 1 | A7H

Description: The setting of the carry bit is complemented; one is changed to zero, and zero is changed to one.

Operation: (C) ← NOT (C)

Example: Set C to one; current setting is unknown.

```
CTO1: CLR C          ;C IS CLEARED TO ZERO
      CPL C          ;C IS SET TO ONE
```

CPL F0 Complement Flag 0

Encoding: | 1 0 0 1 | 0 1 0 1 | 95H

Description: The setting of flag 0 is complemented; one is changed to zero, and zero is changed to one.

Operation: F0 ← NOT (F0)

CPL F1 Complement Flag 1

Encoding: | 1 0 1 1 | 0 1 0 1 | B5H

Description: The setting of flag 1 is complemented; one is changed to zero, and zero is changed to one.

Operation: (F1) ← NOT (F1)

DA A Decimal Adjust Accumulator

Encoding: | 0 1 0 1 | 0 1 1 1 | 57H

Description: The 8-bit accumulator value is adjusted to form two 4-bit Binary Coded Decimal (BCD) digits following the binary addition of BCD numbers. The carry bit C is affected. If the contents of bits 0-3 are greater than nine, or if AC is one, the accumulator is incremented by six.

The four high-order bits are then checked. If bits 4-7 exceed nine, or if C is one, these bits are increased by six. If an overflow occurs, C is set to one.

Example: Assume accumulator contains 10011011.

```
DA A                          ;ACC Adjusted to 00000001
                              ;WITH C SET

C AC 7     4 3     0
0  0 1 0 0 1 1 0 1 1
     0 0 0 0 0 1 1 0          ADD SIX TO BITS 0-7
0  1 1 0 1 0 0 0 0 1
     0 1 1 0                  ADD SIX TO BITS 4-7
1  0 0 0 0 0 0 0 0 1          OVERFLOW TO C
```

DEC A Decrement Accumulator

Encoding: | 0 0 0 0 | 0 1 1 1 | 07H

Description: The contents of the accumulator are decremented by one. The carry flag is not affected.

Operation: (A) ← (A) -1

Example: Decrement contents of external data memory location 63.

```
MOV R0,#3FH                   ;MOVE '3F' HEX TO REG 0
MOVX A, @R0                   ;MOVE CONTENTS OF
                              ;LOCATION 63 TO ACC
DEC A                         ;DECREMENT ACC
MOVX @R0,A                    ;MOVE CONTENTS OF ACC TO
                              ;LOCATION 63 IN EXPANDED
                              ;MEMORY
```

DEC Rr Decrement Register

Encoding: | 1 1 0 0 | 1 r r r | C8H-CFH

Description: The contents of working register 'r' are decremented by one.

Operation: (Rr) ← (Rr) -1 r = 0-7

Example:

```
DECR1: DEC R1                 ;DECREMENT CONTENTS OF REG 1
```

DIS I External Interrupt

Encoding: | 0 0 0 1 | 0 1 0 1 | 15H

Description: External interrupts are disabled. A low signal on the interrupt input pin has no effect.

DIS TCNTI Disable Timer/Counter Interrupt

Encoding: | 0 0 1 1 | 0 1 0 1 | 35H

Description: Timer/counter interrupts are disabled. Any pending timer interrupt request is cleared. The interrupt sequence is not initiated by an overflow, but the timer flag is set and time accumulation continues.

DJNZ R_r, address Decrement Register and Test

Encoding: | 1 1 1 0 | 1 r r r | a_7 a_6 a_5 a_4 | a_3 a_2 a_1 a_0 | E8H-EFH

Description: This is a 2-cycle instruction. Register 'r' is decremented, then tested for zero. If the register contains all zeros, program control falls through to the next instruction. If the register contents are not zero, control jumps to the specified 'address'.

The address in this case must evaluate to 8-bits, that is, the jump must be to a location within the current 256-location page.

Example: (Rr) ← (Rr) -1 r = 0-7
If Rr not 0
(PC_{0-7}) ← addr
Note: A 12-bit address specification does not cause an error if the DJNZ instruction and the jump target are on the same page. If the DJNZ instruction begins in location 255 of a page, it must jump to a target address on the following page.

Example: Increment values in data memory locations 50-54.

```
        MOV R0,#50          ;MOVE '50' DEC TO ADDRESS
                            ;REG 0
        MOV R3,#5           ;MOVE '5' DEC TO COUNTER
                            ;REG 3
INCRT:  INC @R0             ;INCREMENT CONTENTS OF
                            ;LOCATION ADDRESSED BY
                            ;REG 0
        INC R0              ;INCREMENT ADDRESS IN REG 0
        DJNZ R3, INCRT      ;DECREMENT REG 3 — JUMP TO
                            ;'INCRT' IF REG 3 NONZERO
        NEXT —              ;'NEXT' ROUTINE EXECUTED
                            ;IF R3 IS ZERO
```

EN I Enable External Interrupt

Encoding: | 0 0 0 0 | 0 1 0 1 | 05H

Description: External interrupts are enabled. A low signal on the interrupt input pin initiates the interrupt sequence.

EN TCNTI Enable Timer/Counter Interrupt

Encoding: | 0 0 1 0 | 0 1 0 1 | 25H

Description: Timer/counter interrupts are enabled. An overflow of the timer/counter initiates the interrupt sequence.

ENT0 CLK Enable Clock Output

Encoding: | 0 1 1 1 | 0 1 0 1 | 75H

Description: The test 0 pin is enabled to act as the clock output. This function is disabled by a system reset.

Example:
```
EMTST0: ENT0 CLK          ;ENABLE T0 AS CLOCK OUTPUT
```

IN A,Pp Input Port or Data to Accumulator

Encoding: | 0 0 0 0 | 1 0 p p | 09H-0AH

Description: This is a 2-cycle instruction. Data present on port 'p' is transferred (read) to the accumulator.

Operation: (A) ← (Pp)

```
                          p = 1-2
INP12: IN A,P1            ;INPUT PORT 1 CONTENTS TO ACC
       MOV R6,A           ;MOVE ACC CONTENTS TO REG 6
       IN A,P2            ;INPUT PORT 2 CONTENTS TO ACC
       MOV R7,A           ;MOVE ACC CONTENTS TO REG 7
```

INC A Increment Accumulator

Encoding: | 0 0 0 1 | 0 1 1 1 | 17H

Description: The contents of the accumulator are incremented by one. Carry is not affected.

Operation: (A) ← (A) +1

Example: Increment contents of location 100 in external data memory.

```
INCA:  MOV R0,#100      ;MOVE '100' DEC TO ADDRESS REG 0
       MOVX A,@R0       ;MOVE CONTENTS OF LOCATION
                        ;100 TO ACC
       INC A            ;INCREMENT A
       MOVX @R0,A       ;MOVE ACC CONTENTS TO
                        ;LOCATION 101
```

INC R_r Increment Register

Encoding: | 0 0 0 1 | 1 r r r | 18H-1FH

Description: The contents of working register 'r' are incremented by one.

Operation: (Rr) ← (Rr) + 1 r = 0-7

Example:
```
INCR0: INC R0           ;INCREMENT CONTENTS OF REG 0
```

INC @R_i Increment Data Memory Location

Encoding: | 0 0 0 1 | 0 0 0 i | 10H-11H

Description: The contents of the resident data memory location addressed by register 'i' bits 0–5** are incremented by one.

Operation: ((Ri)) ← ((Ri)) + 1 i = 0-1

Example:
```
INCDM: MOV R1,#03FH     ;MOVE ONES TO REG 1
       INC @R1          ;INCREMENT LOCATION 63
```

INS A,BUS* Strobed Input of BUS Data to Accumulator

Encoding: | 0 0 0 0 | 1 0 0 0 | 08H

Description: This is a 2-cycle instruction. Data present on the BUS port is transferred (read) to the accumulator when the RD pulse is dropped. (Refer to section on programming memory expansion for details.)

Operation: (A) ← (BUS)

Example:
```
INPBUS: INS A,BUS       ;INPUT BUS CONTENTS TO ACC
```

* For use with internal program memory ONLY.

** 0–5 in 8048AH/8748H
0–6 in 8049AH/8749H
0–7 in 8050AH

JBb address Jump If Accumulator Bit Is Set

Encoding:

b_2 b_1 b_0 1	0 0 1 0	a_7 a_6 a_5 a_4	a_3 a_2 a_1 a_0

Accumulator Bit	Hex Op Code
0	12
1	32
2	52
3	72
4	92
5	B2
6	D2
7	F2

Description: This is a 2-cycle instruction. Control passes to the specified address if accumulator bit 'b' is set to one.

Operation:

	b = 0–7
(PC_{0-7}) ← addr	If Bb = 1
(PC) = (PC) + 2	If Bb = 0

Example:

```
JB4IS1: JB4 NEXT    ;JUMP TO 'NEXT' ROUTINE
                    ;IF ACC BIT 4 = 1
```

JC address Jump If Carry Is Set

Encoding:

1 1 1 1	0 1 1 0	a_7 a_6 a_5 a_4	a_3 a_2 a_1 a_0	F6H

Description: This is a 2-cycle instruction. Control passes to the specified address if the carry bit is set to one.

Operation:

(PC_{0-7}) ← addr	If C = 1
(PC) = (PC) + 2	If C = 0

Example:

```
JC1: JC OVFLOW      ;JUMP TO 'OVFLOW' ROUTINE
                    ;IF C = 1
```

JF0 address Jump If Flag 0 Is Set

Encoding:

1 0 1 1	0 1 1 0	a_7 a_6 a_5 a_4	a_3 a_2 a_1 a_0	B6H

Description: This is a 2-cycle instruction. Control passes to the specified address if flag 0 is set to one.

Operation:

(PC_{0-7}) ← addr	If F0 = 1
(PC) =(PC) + 2	If F0 = 0

Example:

```
JF0IS1: JF0 TOTAL   ;JUMP TO 'TOTAL' ROUTINE IF F0 = 1
```

JF1 address Jump If Flag 1 Is Set

Encoding:

0 1 1 1	0 1 1 0	a_7 a_6 a_5 a_4	a_3 a_2 a_1 a_0

76H

Description: This is a 2-cycle instruction. Control passes to the specified address if flag 1 is set to one.

Operation: $(PC_{0-7}) \leftarrow$ addr — If F1 = 1
(PC) =(PC + 2) — If F1 = 0

Example:

```
JF1IS1: JF1 FILBUF      ;JUMP TO 'FILBUF'
                        ;ROUTINE IF F1 = 1
```

JMP address Direct Jump within 2K Block

Encoding:

a_{10} a_9 a_8 0	0 1 0 0	a_7 a_6 a_5 a_4	a_3 a_2 a_1 a_0

Page	Hex Op Code
0	04
1	24
2	44
3	64
4	84
5	A4
6	C4
7	E4

Description: This is a 2-cycle instruction. Bits 0–10 of the program counter are replaced with the directly-specified address. The setting of PC bit 11 is determined by the most recent SELECT MB instruction.

Operation: $(PC_{8-10}) \leftarrow$ addr 8–10
$(PC_{0-7}) \leftarrow$ addr 0–7
$(PC_{11}) \leftarrow$ DBF

Example:

```
JMP SUBTOT      ;JUMP TO SUBROUTINE 'SUBTOT'
JMP $-6         ;JUMP TO INSTRUCTION SIX
                ;LOCATIONS BEFORE CURRENT
                ;LOCATION
JMP 2FH         ;JUMP TO ADDRESS '2F' HEX
```

JMPP @A Indirect Jump within Page

Encoding:

1 0 1 1	0 0 1 1

B3H

Description: This is a 2-cycle insruction. The contents of the program memory location pointed to by the accumulator are substituted for the 'page' portion of the program counter (PC bits 0–7).

Operation: $(PC_{0-7}) \leftarrow ((A))$

Example: Assume accumulator contains 0FH.

```
JMPPAG: JMPP @A          ;JUMP TO ADDRESS STORED IN
                         ;LOCATION 15 IN CURRENT PAGE
```

JNC address Jump If Carry Is Not Set

Encoding:

1 1 1 0	0 1 1 0	a_7 a_6 a_5 a_4	a_3 a_2 a_1 a_0	E6H

Description: This is a 2-cycle instruction. Control passes to the specified address if the carry bit is not set, that is, equals zero.

Operation: $(PC_{0-7}) \leftarrow$ addr If C = 0
(PC) = (PC) + 2 If C = 1

Example:

```
JC0: JNC NOVFLO          ;JUMP TO 'NOVFLO' ROUTINE
                         ;IF C = 0
```

JNI address Jump If Interrupt Input Is Low

Encoding:

1 0 0 0	0 1 1 0	a_7 a_6 a_5 a_4	a_3 a_2 a_1 a_0	86H

Description: This is a 2-cycle instruction. Control passes to the specified address if the interrupt input signal is low (= 0), that is, an external interrupt has been signaled. (This signal initiates an interrupt service sequence if the external interrupt is enabled.)

Operation: $(PC_{0-7}) \leftarrow$ addr If I = 0
(PC) = (PC) + 2 If I = 1

Example:

```
LOC 3: JNI EXTINT        ;JUMP TO 'EXTINT' ROUTINE
                         ;IF I = 0
```

JNT0 address Jump If Test 0 is Low

Encoding:

0 0 1 0	0 1 1 0	a_7 a_6 a_5 a_4	a_3 a_2 a_1 a_0	26H

Description: This is a 2-cycle instruction. Control passes to the specified address, if the test 0 signal is low.

Operation: $(PC_{0-7}) \leftarrow$ addr If T0 = 0
(PC) = (PC) + 2 If T0 = 1

Example:

```
JT0LOW: JNT0 60          ;JUMP TO LOCATION 60 DEC
                         ;IF T0 = 0
```

JNT1 address Jump If Test 1 Is Low

Encoding: | 0 1 0 0 | 0 1 1 0 | | a_7 a_6 a_5 a_4 | a_3 a_2 a_1 a_0 | 46H

Description: This is a 2-cycle instruction. Control passes to the specified address, if the test 1 signal is low.

Operation: $(PC_{0-7}) \leftarrow$ addr If T1 = 0
(PC) = (PC) + 2 If T1 = 1

JNZ Address Jump If Accumulator Is Not Zero

Encoding: | 1 0 0 1 | 0 1 1 0 | | a_7 a_6 a_5 a_4 | a_3 a_2 a_1 a_0 | 96H

Description: This is a 2-cycle instruction. Control passes to the specified address if the accumulator contents are nonzero at the time this instruction is executed.

Operation: $(PC_{0-7}) \leftarrow$ addr If A $\neq$ 0
(PC) = (PC) + 2 If A = 0

Example: JACCN0: JNZ 0ABH ;JUMP TO LOCATION 'AB' HEX
;IF ACC VALUE IS NONZERO

JTF address Jump If Timer Flag Is Set

Encoding: | 0 0 0 1 | 0 1 1 0 | | a_7 a_6 a_5 a_4 | a_3 a_2 a_1 a_0 | 16H

Description: This is a 2-cycle instruction. Control passes to the specified address if the timer flag is set to one, that is, the timer/counter register has overflowed. Testing the timer flag resets it to zero. (This overflow initiates an interrupt service sequence if the timer-overflow interrupt is enabled.)

Operation: $(PC_{0-7}) \leftarrow$ addr If TF = 1
(PC) = (PC) + 2 If TF = 0

Example: JTF1: JTF TIMER ;JUMP TO 'TIMER' ROUTINE
;IF TF = 1

JT0 address Jump If Test 0 Is High

Encoding: | 0 0 1 1 | 0 1 1 0 | | a_7 a_6 a_5 a_4 | a_3 a_2 a_1 a_0 | 36H

Description: This is a 2-cycle instruction. Control passes to the specified address if the test 0 signal is high (= 1).

Operation: $(PC_{0-7}) \leftarrow$ addr If T0 = 1
(PC) = (PC) + 2 If T0 = 0

Example: JT0HI: JT0 53 ;JUMP TO LOCATION 53 DEC
;IF T0 = 1

JT1 address Jump If Test 1 Is High

Encoding: | 0 1 0 1 | 0 1 1 0 | | a_7 a_6 a_5 a_4 | a_3 a_2 a_1 a_0 | 56H

Description: This is a 2-cycle instruction. Control passes to the specified address if the test 1 signal is high (= 1).

Operation: $(PC_{0-7}) \leftarrow$ addr — If T1 = 1
(PC) = (PC) + 2 — If T1 = 0

Example:
```
JT1HI: JT1 COUNT        ;JUMP TO 'COUNT' ROUTINE
                        ;IF T1 = 1
```

JZ address Jump If Accumulator Is Zero

Encoding: | 1 1 0 0 | 0 1 1 0 | | a_7 a_6 a_5 a_4 | a_3 a_2 a_1 a_0 | C6H

Description: This is a 2-cycle instruction. Control passes to the specified address if the accumulator contains all zeros at the time this instruction is executed.

Operation: $(PC_{0-7}) \leftarrow$ addr — If A = 0
(PC) = (PC) + 2 — If A $\neq$ 1

Example:
```
JACCO: JZ 0A3H          ;JUMP TO LOCATION 'A3' HEX
                        ;IF ACC VALUE IS ZERO
```

MOV A,#data Move Immediate Data to Accumulator

Encoding: | 0 0 1 0 | 0 0 1 1 | | a_7 a_6 a_5 a_4 | a_3 a_2 a_1 a_0 | 23H

Description: This is a 2-cycle instruction. The 8-bit value specified by 'data' is loaded in the accumulator.

Operation: (A) $\leftarrow$ data

Example:
```
MOV A,#0A3H             ;MOVE 'A3' HEX TO ACC
```

MOV A,PSW Move PSW Contents to Accumulator

Encoding: | 1 1 0 0 | 0 1 1 1 | C7H

Description: The contents of the program status word are moved to the accumulator.

Operation: (A) $\leftarrow$ (PSW)

Example: Jump to 'RB1SET' routine if PSW bank switch, bit 4, is set.
```
BSCHK:  MOV A,PSW       ;MOVE PSW CONTENTS TO ACC
        JB4 RB1SET      ;JUMP TO 'RB1SET' IF ACC BIT 4 = 1
```

MOV A,R_r Move Register Contents to Accumulator

Encoding: | 1 1 1 1 | 1 r r r | F8H–FFH

Description: 8-bits of data are removed from working register 'r' into the accumulator.

Operation: (A) ← (Rr) r = 0–7

Example: MAR: MOV A,R3 ;MOVE CONTENTS OF REG 3 TO ACC

MOV A,@R_i Move Data Memory Contents to Accumulator

Encoding | 1 1 1 1 | 0 0 0 i | F0H–F1H

Description: The contents of the resident data memory location addressed by bits 0–5** of register 'i' are moved to the accumulator. Register 'i' contents are unaffected.

Operation: (A) ← ((Ri)) i = 0–1

Example: Assume R1 contains 00110110.

```
MADM: MOV A,@R1    ;MOVE CONTENTS OF DATA MEM
                   ;LOCATION 54 TO ACC
```

MOV A,T Move Timer/Counter Contents to Accumulator

Encoding: | 0 1 0 0 | 0 0 1 0 | 42H

Description: The contents of the timer/event-counter register are moved to the accumulator.

Operation: (A) ← (T)

Example: Jump to "EXIT" routine when timer reaches '64', that is, when bit 6 set—assuming initialization 64,

```
TIMCHK: MOV A,T     ;MOVE TIMER CONTENTS TO ACC
        JB6 EXIT    ;JUMP TO 'EXIT' IF ACC BIT 6 = 1
```

MOV PSW,A Move Accumulator Contents to PSW

Encoding: | 1 1 0 1 | 0 1 1 1 | D7H

Description: The contents of the accumulator are moved into the progam status word. All condition bits and the stack pointer are affected by this move.

Operation: (PSW) ← (A)

Example: Move up stack pointer by two memory locations, that is, increment the pointer by one.

```
INCPTR: MOV A,PSW    ;MOVE PSW CONTENTS TO ACC
        INC A        ;INCREMENT ACC BY ONE
        MOV PSW,A    ;MOVE ACC CONTENTS TO PSW
```

** 0–5 in 8048AH/8748H
0–6 in 8049AH/8749H
0–7 in 8050AH

MOV R_r,A Move Accumulator Contents to Register

Encoding: | 1 0 1 0 | 1 r r r | A8H-AFH

Description: The contents of the accumulator are moved to register 'r'.

Operation: (Rr) ← (A) r = 0-7

Example:
```
MRA: MOV R0,A          ;MOVE CONTENTS OF ACC TO REG 0
```

MOV R_r,#data Move Immediate Data to Register

Encoding: | 1 0 1 1 | 1 r_2 r_1 r_0 | d_7 d_6 d_5 d_4 | d_3 d_2 d_1 d_0 | B8H-BFH

Description: This is a 2-cycle instruction. The 8-bit value specified by 'data' is moved to register 'r'.

Operation: (Rr) ← data r = 0-7

Examples:
```
MIR4: MOV R4,#HEXTEN     ;THE VALUE OF THE SYMBOL
                         ;'HEXTEN' IS MOVED INTO REG 4
MIR 5: MOV R5,#PI*(R*R)  ;THE VALUE OF THE EXPRESSION
                         ;'PI*(R*R)' IS MOVED INTO REG 5
MIR 6: MOV R6, #0ADH     ;'AD' HEX IS MOVED INTO REG 6
```

MOV @ R_i,A Move Accumulator Contents to Data Memory

Encoding: | 1 0 1 0 | 0 0 0 i | A0H-A1H

Description: The contents of the accumulator are moved to the resident data memory location whose address is specified by bits 0-5** of register 'i'. Register 'i' contents are unaffected.

Operation: ((Ri)) ← (A) i = 0-1

Example: Assume R0 contains 00000111.
```
MDMA: MOV @R0,A          ;MOVE CONTENTS OF ACC TO
                         ;LOCATION 7 (REG 7)
```

MOV @ R_i,#data Move Immediate Data to Data memory

Encoding: | 1 0 1 1 | 0 0 0 i | d_7 d_6 d_5 d_4 | d_3 d_2 d_1 d_0 | B0H-B1H

Description: This is a 2-cycle instruction. The 8-bit value specified by 'data' is moved to the resident data memory location addressed by register 'i', bits 0-5**.

Operation: ((Ri)) ← data i = 0-1

Examples: Move the hexadecimal value AC3F to locations 62-63.
```
MIDM: MOV R0,#62         ;MOVE '62' DEC TO ADDR REG 0
      MOV @R0,#0ACH      ;MOVE 'AC' HEX TO LOCATION 62
      INC R0             ;INCREMENT REG 0 to '63'
      MOV @R0,#3FH       ;MOVE '3F' HEX TO LOCATION 63
```

** 0-5 in 8048AH/8748H
0-6 in 8049AH/8749H
0-7 in 8050AH

MOV T,A Move Accumulator Contents to Timer/Counter

Encoding: | 0 1 1 0 | 0 0 1 0 | 62H

Description: The contents of the accumulator are moved to the timer/event-counter register.

Operation: (T) ← (A)

Example: Initialize and start event counter.

```
INITEC: CLR A          ;CLEAR ACC TO ZEROS
        MOV T,A        ;MOVE ZEROS TO EVENT COUNTER
        START CNT      ;START COUNTER
```

MOVD A,Pp Move Port 4-7 Data to Accumulator

Encoding: | 0 0 0 0 | 1 1 p p | 0CH-0FH

Description: This is a 2-cycle instruction. Data on 8243 port 'p' is moved (read) to accumulator bits 0-3. Accumulator bits 4-7 are zeroed.

Operation: (0-3) ← (Pp) p = 4-7
(4-7) ← 0

Note: Bits 0-7 of the opcode are used to represent ports 4-7. If you are coding in binary rather than assembly language, the mapping is as follows:

Bits 1 0	Port
0 0	4
0 1	5
1 0	6
1 1	7

Example:

```
INPPT5: MOVD A,P5      ;MOVE PORT 5 DATA TO ACC
                       ;BITS 0-3, ZERO ACC BITS 4-7
```

MOVD Pp,A Move Accumulator Data to Port 4-7

Encoding: | 0 0 1 1 | 1 1 p p | 3CH-3FH

Description: This is a 2-cycle instruction. Data in accumulator bits 0-3 is moved (written) to 8243 port 'p'. Accumulator bits 4-7 are unaffected. (See NOTE above regarding port mapping.)

Operation: (Pp) ← (A_{0-3}) P = 4-7

Example: Move data in accumulator to ports 4 and 5.

```
OUTP45: MOVD P4,A      ;MOVE ACC BITS 0-3 TO PORT 4
        SWAP A         ;EXCHANGE ACC BITS 0-3 and 4-7
        MOVD P5,A      ;MOVE ACC BITS 0-3 TO PORT 5
```

MOVP A,@A Move Current Page Data to Accumulator

Encoding: | 1 0 1 0 | 0 0 1 1 | A3H

Description: The contents of the program memory location addressed by the accumulator are moved to the accumulator. Only bits 0–7 of the program counter are affected, limiting the program memory reference to the current page. The program counter is restored *following* this operation.

Operation: $(PC_{0-7}) \leftarrow (A)$
$(A) \leftarrow ((PC))$
Note: This is a 1-byte, 2-cycle instruction. If it appears in location 255 of a program memory page, @A addresses a location in the *following* page.

Example:
```
MOV128: MOV A,#128      ;MOVE '128' DEC TO ACC
        MOVP A,@A       ;CONTENTS OF 129th LOCATION IN
                        ;CURRENT PAGE ARE MOVED TO ACC
```

MOVP3 A,@A Move Page 3 Data to Accumulator

Encoding: | 1 1 1 0 | 0 0 1 1 | E3H

Description: This is a 2-cycle instruction. The contents of the program memory location (within page 3) addressed by the accumulator are moved to the accumulator. The program counter is restored following this operation.

Operation: $(PC_{0-7}) \leftarrow (A)$
$(PC_{8-11}) \leftarrow 0011$
$(A) \leftarrow ((PC))$

Example: Look up ASCII equivalent of hexadecimal code in table contained at the beginning of page 3. Note that ASCII characters are designated by a 7-bit code; the eighth bit is always reset.

```
TABSCH: MOV A,#0B8H     ;MOVE 'B8' HEX TO ACC (10111000)
        ANL A,#7FH      ;LOGICAL AND ACC TO MASK BIT
                        ;7 (00111000)
        MOVP3 A,@A      ;MOVE CONTENTS OF LOCATION '38'
                        ;HEX IN PAGE 3 TO ACC (ASCII '8')
```

Access contents of location in page 3 labelled TAB1.
Assume current program location is not in page 3.

```
TABSCH: MOV A,#LOW TAB 1 ;ISOLATE BITS 0-7 OF LABEL
                         ;ADDRESS VALUE
        MOVP3 A,@A       ;MOVE CONTENTS OF PAGE 3
                         ;LOCATION LABELED 'TAB1' TO ACC
```

MOVX A,@R_i Move External-Data-Memory Contents to Accumulator

Encoding: | 1 0 0 0 | 0 0 0 i | 80H-81H

Description: This is a 2-cycle instruction. The contents of the external data memory location addressed by register 'i' are moved to the accumulator. Register 'i' contents are unaffected. A read pulse is generated.

Operation: (A) ← ((Ri)) i = 0-1

Example: Assume R1 contains 01110110.

```
MAXDM:  MOVX A,@R1      ;MOVE CONTENTS OF LOCATION
                        ;118 TO ACC
```

MOVX @R_i,A Move Accumulator Contents to External Data Memory

Encoding: | 1 0 0 1 | 0 0 0 i | 90H-91H

Description: This is a 2-cycle instruction. The contents of the accumulator are moved to the external data memory location addressed by register 'i'. Register 'i' contents are unaffected. A write pulse is generated.

Operation: ((Ri)) ← A i = 0-1

Example: Assume R0 contains 11000111.

```
MXDMA:  MOVX @R0,A      ;MOVE CONTENTS OF ACC TO
                        ;LOCATION 199 IN EXPANDED
                        ;DATA MEMORY
```

NOP The NOP Instruction

Encoding: | 0 0 0 0 | 0 0 0 0 | 00H

Description: No operation is performed. Execution continues with the following instruction.

ORL A,R_r Logical OR Accumulator With Register Mask

Encoding: | 0 1 0 0 | 1 r r r | 48H-4FH

Description: Data in the accumulator is logically ORed with the mask contained in working register 'r'.

Operation: (A) ← (A) OR (Rr) r = 0-7

Example:

```
ORREG:  ORL A,R4        ;'OR' ACC CONTENTS WITH
                        ;MASK IN REG 4
```

ORL A,@R_i Logical OR Accumulator With Memory Mask

Encoding: | 0 1 0 0 | 0 0 0 i | 40H-41H

Description: Data in the accumulator is logically ORed with the mask contained in the resident data memory location referenced by register "i", bits 0–5**.

Operation: (A) ← (A) OR ((Ri)) i = 0-1

Example:
```
ORDM: MOV R0,#3FH     ;MOVE '3F' HEX TO REG 0
      ORL A,@R0       ;'OR' AC CONTENTS WITH MASK
                      ;IN LOCATION 63
```

ORL A,#data Logical OR Accumulator With Immediate Mask

Encoding: | 0 1 0 0 | 0 0 1 1 | d_7 d_6 d_5 d_4 | d_3 d_2 d_1 d_0 | 43H

Description: This is a 2-cycle instruction. Data in the accumulator is logically ORed with an immediately-specified mask.

Operation: (A) ← (A) OR data

Example:
```
ORID: ORL A,#'X'      ;'OR' ACC CONTENTS WITH MASK
                      ;01011000 (ASCII VALUE OF 'X')
```

ORL BUS,#data* Logical OR BUS With Immediate Mask

Encoding: | 1 0 0 0 | 1 0 0 0 | d_7 d_6 d_5 d_4 | d_3 d_2 d_1 d_0 | 88H

Description: This is a 2-cycle instruction. Data on the BUS port is logically ORed with an immediately-specified mask. This instruction assumes prior specification on an 'OUTL BUS,A' instruction.

Operation: (BUS) ← (BUS) OR data

Example:
```
ORBUS: ORL BUS,#HEXMSK  :'OR' BUS CONTENTS WITH MASK
                        ;EQUAL VALUE OF SYMBOL 'HEXMSK'
```

ORL Pp, #data Logical OR Port 1 or 2 With Immediate Mask

Encoding: | 1 0 0 0 | 1 0 p p | d_7 d_6 d_5 d_4 | d_3 d_2 d_1 d_0 | 89H-8AH

Description: This is a 2-cycle instruction. Data on port 'p' is logically ORed with an immediately-specified mask.

Operation: (Pp) ← (Pp) OR data p = 1-2

Example:
```
ORP1: ORL P1, #0FFH   ;'OR' PORT 1 CONTENTS WITH MASK
                      ;'FF' HEX (SET PORT 1 TO ALL ONES)
```

* For use with internal program memory ONLY.
** 0–5 in 8048AH/8748H
0–6 in 8049AH/8749H
0–7 in 8050AH

ORLD Pp,A Logical OR Port 4-7 With Accumulator Mask

Encoding: | 1 0 0 0 | 1 1 p p | 8CH-8FH

Description: This is a 2-cycle instruction. Data on port 'p' is logically ORed with the digit mask contained in accumulator bits 0-3.

Operation: (Pp) ← (Pp) OR (A_{0-3}) p = 4-7

Example:
```
ORP7: ORLD P7,A      ;'OR' PORT 7 CONTENTS WITH ACC
                     ;BITS 0-3
```

OUTL BUS,A* Output Accumulator Data to BUS

Encoding: | 0 0 0 0 | 0 0 1 0 | 02H

Description: This is a 2-cycle instruction. Data residing in the accumulator is transferred (written) to the BUS port and latched. The latched data remains valid until altered by another OUTL instruction. Any other instruction requiring use of the BUS port (except INS) destroys the contents of the BUS latch. This includes expanded memory operations (such as the MOVX instruction). Logical operations on BUS data (AND, OR) assume the OUTL BUS,A instruction has been issued previously.

Operation: (BUS) ← (A)

Example:
```
OUTLBP: OUTL BUS, A      ;OUTPUT ACC CONTENTS TO BUS
```

OUTL Pp,A Output Accumulator Data to Port 1 or 2

Encoding: | 0 0 1 1 | 1 0 p p | 39H-3AH

Description: This is a 2-cycle instruction. Data residing in the accumulator is transferred (written) to port 'p' and latched.

Operation: (Pp) ← (A) p = 1-2

Example:
```
OUTLP: MOV A,R7      ;MOVE REG 7 CONTENTS TO ACC
       OUTL P2,A     ;OUTPUT ACC CONTENTS TO PORT 2
       MOV A, R6     ;MOV REG 6 CONTENTS TO ACC
       OUTL P1,A     ;OUTPUT ACC CONTENTS TO PORT 1
```

* For use with internal program memory ONLY.

RET Return Without PSW Restore

Encoding: | 1 0 0 0 | 0 0 1 1 | 83H

Description: This is a 2-cycle instruction. The stack pointer (PSW bits 0-2) is decremented. The program counter is then restored from the stack. PSW bits 4-7 are not restored.

Operation: (SP) ← (SP)-1
(PC) ← ((SP))

RETR Return with PSW Restore

Encoding: | 1 0 0 1 | 0 0 1 1 | 93H

Description: This is a 2-cycle instruction. The stack pointer is decremented. The program counter and bits 4-7 of the PSW are then restored from the stack. Note that RETR should be used to return from an interrupt, but should not be used within the interrupt service routine as it signals the end of an interrupt routine by resetting the Interrupt in Progress flip-flop.

Operation: (SP) ← (SP)-1
(PC) ← ((SP))
(PSW 4-7) ← ((SP))

RL A Rotate Left without Carry

Encoding: | 1 1 1 0 | 0 1 1 1 | E7H

Description: The contents of the accumulator are rotated left one bit. Bit 7 is rotated into the bit 0 position.

Operation: (An + 1) ← (An)
(A0) ← (A7) n = 0-6

Example: Assume accumulator contains 10110001.

```
RLNC: RL A          ;NEW ACC CONTENTS ARE 01100011
```

RLC A Rotate Left through Carry

Encoding: | 1 1 1 1 | 0 1 1 1 | F7H

Description: The contents of the accumulator are rotated left one bit. Bit 7 replaces the carry bit; the carry bit is rotated into the bit 0 position.

Operation: (An + 1) ← (An)
n = 0-6
(A0) ← (C)
(C) ← (A7)

Example: Assume accumulator contains a 'signed' number; isolate sign without changing value.

```
RLTC: CLR C         ;CLEAR CARRY TO ZERO
      RLC A         ;ROTATE ACC LEFT, SIGN
                    ;BIT(7) IS PLACED IN CARRY
      RR A          ;ROTATE ACC RIGHT — VALUE
                    ;(BITS 0-6) IS RESTORED,
                    ;CARRY UNCHANGED, BIT 7
                    ;IS ZERO
```

RR A Rotate Right without Carry

Encoding: | 0 1 1 1 | 0 1 1 1 | 77H

Description: The contents of the accumulator are rotated right one bit. Bit 0 is rotated into the bit 7 position.

Operation: (An) ← (An + 1) n = 0-6
(A7) ← (A0)

Example: Assume accumulator contains 10110001.

```
RRNC: RR A          ;NEW ACC CONTENTS ARE 11011000
```

RRC A Rotate Right through Carry

Encoding: | 0 1 1 0 | 0 1 1 1 | 67H

Description: The contents of the accumulator are rotated right one bit. Bit 0 replaces the carry bit; the carry bit is rotated into the bit 7 position.

Operation: (An) ← (An + 1) n = 0-6
(A7) ← (C)
(C) ← (A_0)

Example: Assume carry is not set and accumulator contains 10110001.

```
RRTC: RRC A          ;CARRY IS SET AND ACC
                     ;CONTAINS 01011000
```

SEL MB0 Select Memory Bank 0

Encoding: | 1 1 1 0 | 0 1 0 1 | E5H

Description: PC bit 11 is set to zero on next JMP or CALL instruction. All references to program memory addresses fall within the range 0-2047.

Operation: (DBF) ← 0

Example: Assume program counter contains 834 Hex.

```
SEL MB0              ;SELECT MEMORY BANK 0
JMP $+20             ;JUMP TO LOCATION 58 HEX
```

SEL MB1 Select Memory Bank 1

Encoding: | 1 1 1 1 | 0 1 0 1 | F5H

Description: PC bit 11 is set to one on next JMP or CALL instruction. All references to program memory addresses fall within the range 2048-4095.

Operation: (DBF) ← 1

SEL RB0 Select Register Bank 0

Encoding: | 1 1 0 0 | 0 1 0 1 | C5H

Description: PSW bit 4 is set to zero. References to working registers 0-7 address data memory locations 0-7. This is the recommended setting for normal program execution.

Operation: (BS) ← 0

SEL RB1 Select Register Bank 1

Encoding: | 1 1 0 1 | 0 1 0 1 | D5H

Description: PSW bit 4 is set to one. References to working registers 0-7 address data memory locations 24-31. This is the recommended setting for interrupt service routines, since locations 0-7 are left intact. The setting of PSW bit 4 in effect at the time of an interrupt is restored by the RETR instruction when the interrupt service routine is completed.

Operation: (BS) ← 1

Example: Assume an external interrupt has occurred, control has passed to program memory location 3, and PSW bit 4 was zero before the interrupt.

Operation:

```
LOC3:  JNI INIT          ;JUMP TO ROUTINE 'INIT' IF
                         ;INTERRUPT INPUT IS ZERO
INIT:  MOV R7,A          ;MOVE ACC CONTENTS TO
                         ;LOCATION 7
       SEL RB1           ;SELECT REG BANK 1
       MOV R7,#0FAH      ;MOVE 'FA' HEX TO LOCATION 31
       .
       .
       .
       SEL RB0           ;SELECT REG BANK 0
       MOV A,R7          ;RESTORE ACC FROM LOCATION 7
       RETR              ;RETURN — RESTORE PC AND PSW
```

STOP TCNT Stop Timer/Event-Counter

Encoding: | 0 1 1 0 | 0 1 0 1 | 65H

Description: This instruction is used to stop both time accumulation and event counting.

Example: Disable interrupt, but jump to interrupt routine after eight overflows and stop timer. Count overflows in register 7.

```
START:  DIS TCNTI          ;DISABLE TIMER INTERRUPT
        CLR A              ;CLEAR ACC TO ZEROS
        MOV T,A            ;MOVE ZEROS TO TIMER
        MOV R7,A           ;MOVE ZEROS TO REG 7
        STRT T             ;START TIMER
MAIN:   JTF COUNT          ;JUMP TO ROUTINE 'COUNT'
                           ;IF TF = 1 AND CLEAR TIMER FLAG
        JMP MAIN           ;CLOSE LOOP
COUNT:  INC R7             ;INCREMENT REG 7
        MOV A,R7           ;MOVE REG 7 CONTENTS TO ACC
        JB3 INT            ;JUMP TO ROUTINE 'INT' IF ACC
                           ;BIT 3 IS SET (REG 7 = 8)
        JMP MAIN           ;OTHERWISE RETURN TO ROUTINE
                           ;MAIN
        .
        .
        .
INT:    STOP TCNT          ;STOP TIMER
        JMP 7H             ;JUMP TO LOCATION 7 (TIMER)
                           ;INTERRUPT ROUTINE
```

STRT CNT Start Event Conter

Encoding: | 0 1 0 0 | 0 1 0 1 | 45H

Description: The test 1 (T1) pin is enabled as the event-counter input and the counter is started. The event-counter register is incremented with each high-to-low transition on the T1 pin.

Example: Initialize and start event counter. Assume overflow is desired with first T1 input.

```
STARTC: EN TCNTI           ;ENABLE COUNTER INTERRUPT
        MOV A,#0FFH        ;MOVE 'FF'HEX (ONES) TO ACC
        MOV T,A            ;MOVES ONES TO COUNTER
        STRT CNT           ;ENABLE T1 AS COUNTER
                           ;INPUT AND START
```

STRT T Start Timer

Encoding: | 0 1 0 1 | 0 1 0 1 | 55H

Description: Timer accumulation is initiated in the timer register. The register is incremented every 32 instruction cycles. The prescaler which counts the 32 cycles is cleared but the timer register is not.

Example: Initialize and start timer.

```
STARTT: CLR A          ;CLEAR ACC TO ZEROS
        MOV T,A        ;MOVE ZEROS TO TIMER
        EN TCNTI       ;ENABLE TIMER INTERRUPT
        STRT T         ;START TIMER
```

SWAP A Swap Nibbles within Accumulator

Encoding: | 0 1 0 0 | 0 1 1 1 | 47H

Description: Bits 0-3 of the accumulator are swapped with bits 4-7 of the accumulator.

Operation: $(A_{4-7}) \rightleftarrows (A_{0-3})$

Example: Pack bits 0-3 of locations 50-51 into location 50.

```
PCKDIG: MOV R0, #50      ;MOVE '50' DEC TO REG 0
        MOV R1, #51      ;MOVE '51' DEC TO REG 1
        XCHD A,@R0       ;EXCHANGE BITS 0-3 OF ACC
                         ;AND LOCATION 50
        SWAP A           ;SWAP BITS 0-3 AND 4-7 OF ACC
        XCHD A,@R1       ;EXCHANGE BITS 0-3 OF ACC AND
                         ;LOCATION 51
        MOV @R0,A        ;MOVE CONTENTS OF ACC TO
                         ;LOCATION 50
```

XCH A,R_r Exchange Accumulator-Register Contents

Encoding: | 0 0 1 0 | 1 r r r | 28H-2FH

Description: The contents of the accumulator and the contents of working register 'r' are exchanged.

Operation: $(A) \rightleftarrows (Rr)$ r = 0-7

Example: Move PSW contents to Reg 7 without losing accumulator contents.

```
XCHAR7: XCH A,R7        ;EXCHANGE CONTENTS OF REG 7
                        ;AND ACC
        MOV A, PSW      ;MOVE PSW CONTENTS TO ACC
        XCH A,R7        ;EXCHANGE CONTENTS OF REG 7
                        'AND ACC AGAIN
```

XCH A,@R_i Exchange Accumulator and Data Memory Contents

Encoding: | 0 0 1 0 | 0 0 0 i | 20H-21H

Description: The contents of the accumulator and the contents of the resident data memory location addressed by bits 0-5** of register 'i' are exchanged. Register 'i' contents are unaffected.

Operation: (A) ⇆ ((Ri)) i = 0-1

Example: Decrement contents of location 52.

```
DEC52: MOV R0,#52      ;MOVE '52' DEC TO ADDRESS REG 0
       XCH A,@R0       ;EXCHANGE CONTENTS OF ACC
                       ;AND LOCATION 52
       DEC A           ;DECREMENT ACC CONTENTS
       XCH A,@R0       ;EXCHANGE CONTENTS OF ACC
                       ;AND LOCATION 52 AGAIN
```

XCHD A,@R_i Exchange Accumulator and Data Memory 4-Bit Data

Encoding: | 0 0 1 1 | 0 0 0 i | 30H-31H

Description: This instruction exchanges bits 0-3 of the accumulator with bits 0-3 of the data memory location addressed by bits 0-5** of register 'i'. Bits 4-7 of the accumulator, bits 4-7 of the data memory location, and the contents of register 'i' are unaffected.

Operation: (A_{0-3}) ⇆ ((Ri0-3)) i = 0-1

Example: Assume program counter contents have been stacked in locations 22-23.

```
XCHNIB: MOV R0,#23      ;MOVE '23' DEC TO REG 0
        CLR A           ;CLEAR ACC TO ZEROS
        XCHD A,@R0      ;EXCHANGE BITS 0-3 OF ACC AND
                        ;LOCATION 23 (BTS 8-11 OF PC ARE
                        ;ZEROED, ADDRESS REFERS
                        :TO PAGE 0)
```

XRL A,R_r Logical XOR Accumulator With Register Mask

Encoding: | 1 1 0 1 | 1 r r r | D8H-DFH

Description: Data in the accumulator is EXCLUSIVE ORed with the mask contained in working register 'r'.

Operation: (A) ← (A) XOR (Rr) r = 0-7

Example:

```
XORREG: XRL A,R5        ;'XOR' ACC CONTENTS WITH
                        ;MASK IN REG 5
```

** 0-5 in 8048AH/8748H
0-6 in 8049AH/8749H
0-7 in 8050AH

XRL A,@R_i Logical XOR Accumulator With Memory Mask

Encoding: | 1 1 0 1 | 0 0 0 i | D0H-D1H

Description: Data in the accumulator is EXCLUSIVE ORed with the mask contained in the data memory location addressed by register 'i', bits 0–5.**

Operation: (A) ← (A) XOR ((Ri)) i = 0-1

Example:
```
XORDM:  MOV R1,#20H     ;MOVE '20' HEX TO REG 1
        XRL A,@R1       ;'XOR' ACC CONTENTS WITH MASK
                        ;IN LOCATION 32
```

XRL A,#data Logical XOR Accumulator With Immediate Mask

Encoding: | 1 1 0 1 | 0 0 1 1 | d_7 d_6 d_5 d_4 | d_3 d_2 d_1 d_0 | D3H

Description: This is a 2-cycle instruction. Data in the accumulator is EXCLUSIVE ORed with an immediately-specified mask.

Operation: (A) ← (A) XOR data

Example:
```
XORID:  XOR A,#HEXTEN   ;XOR CONTENTS OF ACC WITH MASK
                        ;EQUAL VALUE OF SYMBOL 'HEXTEN'
```

** 0–5 in 8048AH/8748H
0–6 in 8049AH/8749H
0–7 in 8050AH

XRL A,@Ri Logical XOR Accumulator With Memory Mask

Encoding: 1 1 0 1 | 0 0 0 i D0H-D1H

Description: Data in the accumulator is EXCLUSIVE ORed with the mask contained in the data memory location addressed by register i. bits 0–5.

Operation: (A) ← (A) XOR ((Ri)) i = 0-1

Example: XORDM: MOV R1,#20H
XRL A,@R1

;MOVE 20' HEX TO REG 1.
;XOR ACC CONTENTS WITH MASK
;IN LOCATION 32

XRL A,#data Logical XOR Accumulator With Immediate Mask

Encoding: 1 1 0 1 | 0 0 1 1 d_7 d_6 d_5 d_4 | d_3 d_2 d_1 d_0 D3H

Description: This is a 2-cycle instruction. Data in the accumulator is EXCLUSIVE ORed with an immediately-specified mask

Operation: (A) ← (A) XOR data

Example: XORIS: XOR A,#NEXTEN

;XOR CONTENTS OF ACC WITH MASK
;EQUAL VALUE OF SYMBOL 'NEXTEN'

0–3 in 8048AH/8748H
0–6 in 8048AH/8748H
0–7 in 8050AH

MCS®-48 Data Sheets

4

8243
MCS®-48 INPUT/OUTPUT EXPANDER

■ 0°C TO 70°C Operation

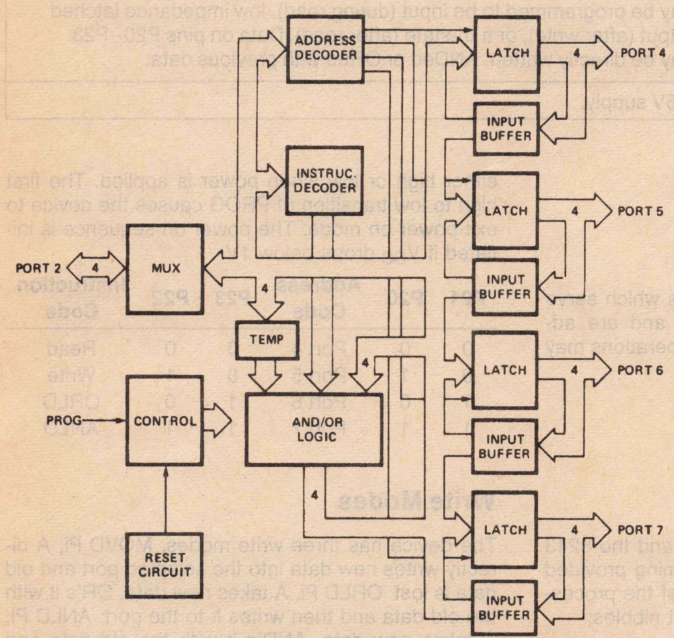


Figure 1. 8243 Block Diagram

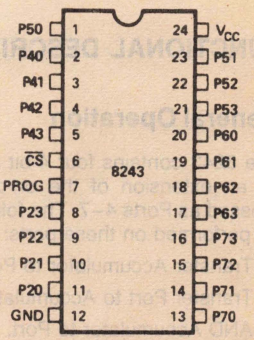


Figure 2. 8243 Pin Configuration

October 1986
Order Number: 270161-001

Table 1. Pin Description

Symbol	Pin No.	Function
PROG	7	Clock Input. A high to low transition on PROG signifies that address and control are available on P20–P23, and a low to high transition signifies that data is available on P20–P23.
$\overline{CS}$	6	Chip Select Input. A high on CS inhibits any change of output or internal status.
P20–P23	11–8	Four (4) bit bi-directional port contains the address and control bits on a high to low transition of PROG. During a low to high transition, P2 contains the data for a selected output port if a write operation, or the data from a selected port before the low to high transition if a read operation.
GND	12	0V supply.
P40–P43	2–5	Four (4) bit bi-directional I/O ports.
P50–P53 P60–P63 P70–P73	1, 23–21 20–17 13–16	May be programmed to be input (during read), low impedance latched output (after write), or a tri-state (after read). Data on pins P20–P23 may be directly written, ANDed or ORed with previous data.
V_{CC}	24	+5V supply.

FUNCTIONAL DESCRIPTION

General Operation

The 8243 contains four 4-bit I/O ports which serve as an extension of the on-chip I/O and are addressed as Ports 4–7. The following operations may be performed on these ports:

- Transfer Accumulator to Port.
- Transfer Port to Accumulator.
- AND Accumulator to Port.
- OR Accumulator to Port.

All communication between the 8048 and the 8243 occurs over Port 2 (P20–P23) with timing provided by an output pulse on the PROG pin of the processor. Each transfer consists of two 4-bit nibbles:

The first containing the "op code" and port address and the second containing the actual 4-bits of data. A high to low transition of the PROG line indicates that address is present while a low to high transition indicates the presence of data. Additional 8243's may be added to the 4-bit bus and chip selected using additional output lines from the 8048/8748/8035.

Power On Initialization

Initial application of power to the device forces input/output Ports 4, 5, 6, and 7 to the tri-state and Port 2 to the input mode. The PROG pin may be either high or low when power is applied. The first high to low transition of PROG causes the device to exit power on mode. The power on sequence is initiated if V_{CC} drops below 1V.

P21	P20	Address Code	P23	P22	Instruction Code
0	0	Port 4	0	0	Read
0	1	Port 5	0	1	Write
1	0	Port 6	1	0	ORLD
1	1	Port 7	1	1	ANLD

Write Modes

The device has three write modes. MOVD Pi, A directly writes new data into the selected port and old data is lost. ORLD Pi, A takes new data, OR's it with the old data and then writes it to the port. ANLD Pi, A takes new data, AND's it with the old data and then writes it to the port. Operation code and port address are latched from the input Port 2 on the high to low transition of the PROG pin. On the low to high transition of PROG data on Port 2 is transferred to the logic block of the specified output port.

After the logic manipulation is performed, the data is latched and outputed. The old data remains latched until new valid outputs are entered.

Read Mode

The device has one read mode. The operation code and port address are latched from the input Port 2

on the high to low transition of the PROG pin. As soon as the read operation and port address are decoded, the appropriate outputs are tri-stated, and the input buffers switched on. The read operation is terminated by a low to high transition of the PROG pin. The port (4, 5, 6 or 7) that was selected is switched to the tri-stated mode while Port 2 is returned to the input mode.

Normally, a port will be in an output (write mode) or input (read mode). If modes are changed during operation, the first read following a write should be ignored; all following reads are valid. This is to allow the external driver on the port to settle after the first read instruction removes the low impedance drive from the 8243 output. A read of any port will leave that port in a high impedance state.

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias .....0°C to 70°C
Storage Temperature .....−65°C to +150°C
Voltage on Any Pin
with Respect to Ground.....−0.5V to +7V
Power Dissipation .....1 Watt

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS T_A = 0°C to 70°C, V_{CC} = 5V ± 10%

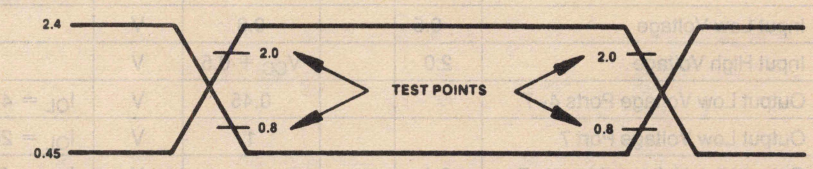
Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
V_{IL}	Input Low Voltage	−0.5		0.8	V	
V_{IH}	Input High Voltage	2.0		V_{CC} + 0.5	V	
V_{OL1}	Output Low Voltage Ports 4–7			0.45	V	I_{OL} = 4.5 mA*
V_{OL2}	Output Low Voltage Port 7			1	V	I_{OL} = 20 mA
V_{OH1}	Output High Voltage Ports 4–7	2.4			V	I_{OH} = 240 μA
I_{IL1}	Input Leakage Ports 4–7	−10		20	μA	V_{in} = V_{CC} to 0V
I_{IL2}	Input Leakage Port 2, CS, PROG	−10		10	μA	V_{in} = V_{CC} to 0V
V_{OL3}	Output Low Voltage Port 2			0.45	V	I_{OL} = 0.6 mA
I_{CC}	V_{CC} Supply Current		10	20	mA	(Note 1)
V_{OH2}	Output Voltage Port 2	2.4				I_{OH} = 100 μA
I_{OL}	Sum of All I_{OL} From 16 Outputs			72	mA	4.5 mA Each Pin

*Refer to Figure 3 for additional sink current capability.

A.C. CHARACTERISTICS $T_A = 0°C$ to $70°C$, $V_{CC} = 5V \pm 10\%$

Symbol	Parameter	Min	Max	Units	Test Conditions
t_A	Code Valid before PROG	50		ns	80 pF Load
t_B	Code Valid after PROG	60		ns	20 pF Load
t_C	Data Valid before PROG	200		ns	80 pF Load
t_D	Data Valid after PROG	20		ns	20 pF Load
t_H	Floating after PROG	0	150	ns	20 pF Load
t_K	PROG Negative Pulse Width	700		ns	
t_{CS}	CS Valid before/after PROG	50		ns	
t_{PO}	Ports 4–7 Valid after PROG		700	ns	100 pF Load
t_{LP1}	Ports 4–7 Valid before/after PROG	100		ns	
t_{ACC}	Port 2 Valid after PROG		650	ns	80 pF Load

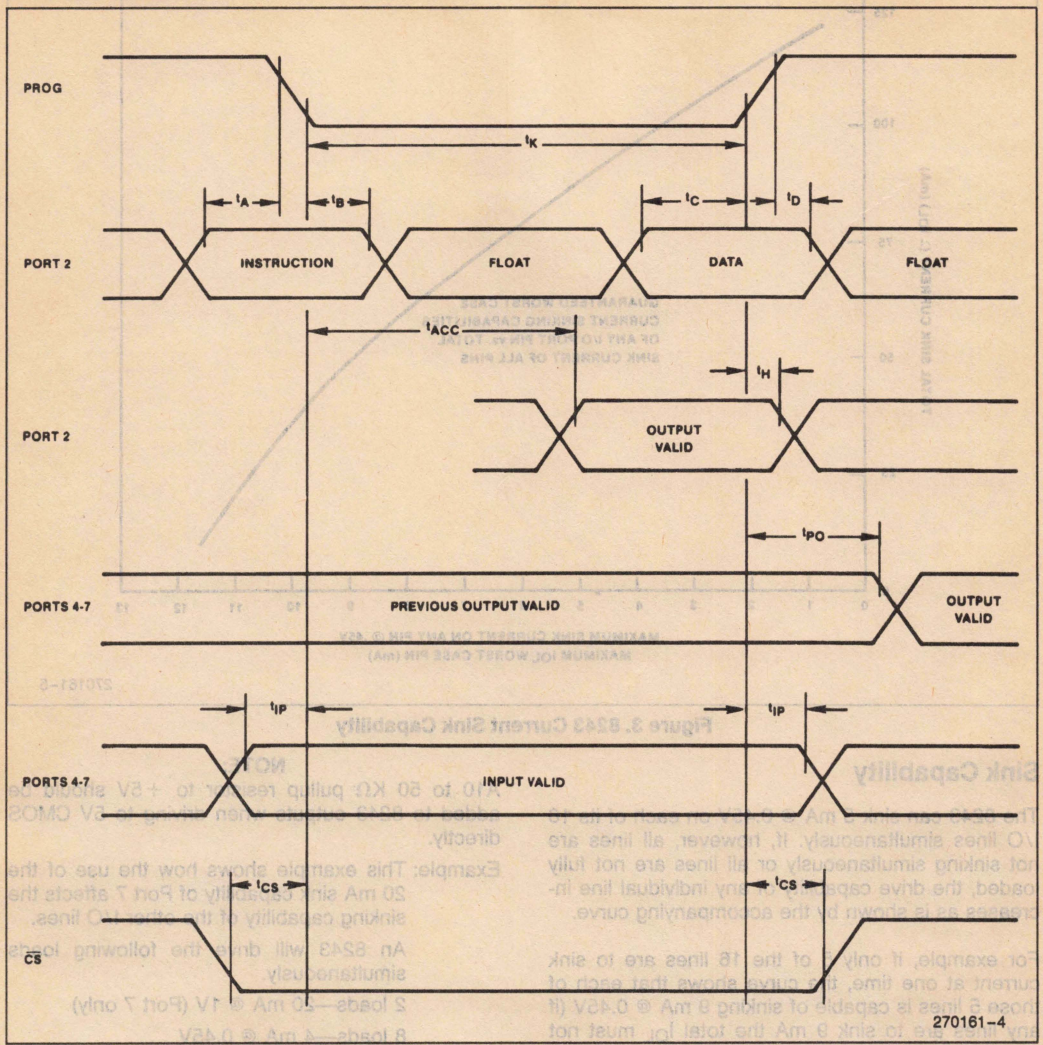
NOTE:
1. I_{CC} (−40°C to 85°C EXPRESS options) 15 mA typical/25 mA maximum.



270161-3

A.C. Testing: Inputs are driven at 2.4V for a Logic "1" and 0.45V for a Logic "0". Output timing measurements are made at 2.0V for Logic "1" and 0.8V for a Logic "0".

WAVEFORMS


PROG
t_K
t_A
t_B
t_C
t_D
PORT 2
INSTRUCTION
FLOAT
DATA
FLOAT
t_{ACC}
t_H
PORT 2
OUTPUT VALID
t_{PO}
PORTS 4-7
PREVIOUS OUTPUT VALID
OUTPUT VALID
t_{IP}
t_{IP}
PORTS 4-7
INPUT VALID
t_{CS}
t_{CS}
$\overline{CS}$
270161-4

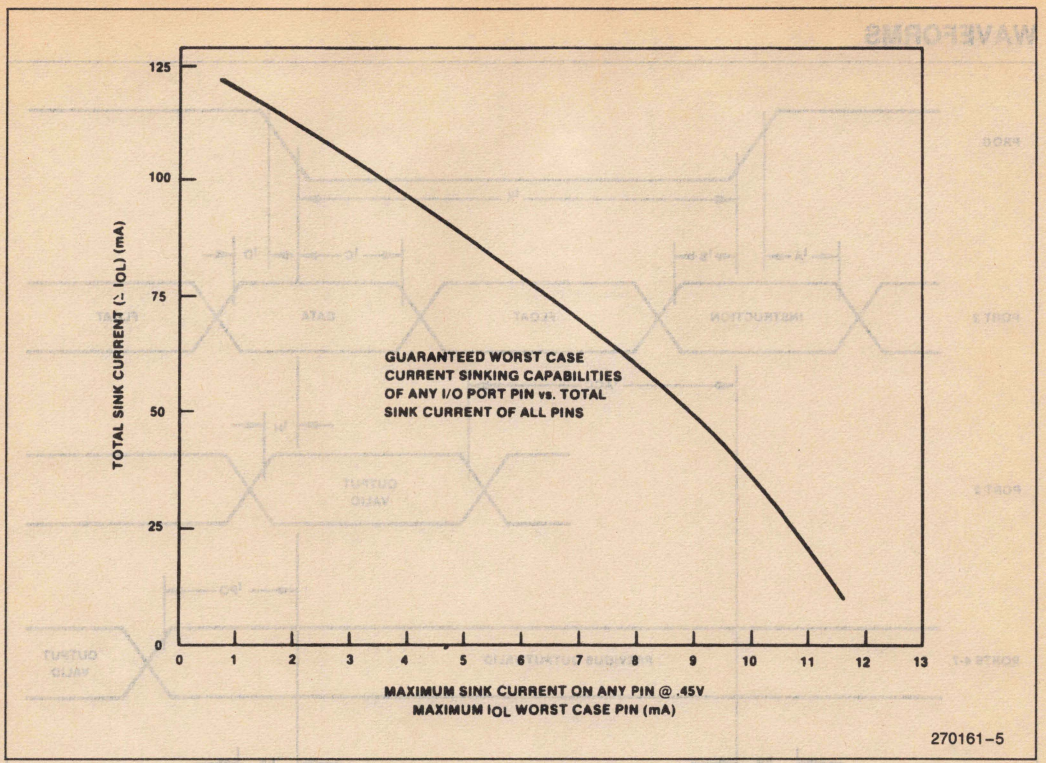


Figure 3. 8243 Current Sink Capability

Sink Capability

The 8243 can sink 5 mA @ 0.45V on each of its 16 I/O lines simultaneously. If, however, all lines are not sinking simultaneously or all lines are not fully loaded, the drive capability of any individual line increases as is shown by the accompanying curve.

For example, if only 5 of the 16 lines are to sink current at one time, the curve shows that each of those 5 lines is capable of sinking 9 mA @ 0.45V (if any lines are to sink 9 mA the total I_{OL} must not exceed 45 mA or five 9 mA loads).

Example: How many pins can drive 5 TTL loads (1.6 mA) assuming remaining pins are unloaded?

$I_{OL} = 5 \times 1.6\ mA = 8\ mA$

$\epsilon I_{OL} = 60$ mA from curve

pins $= 60\ mA \div 8\ mA/pin = 7.5 = 7$

In this case, 7 lines can sink 8 mA for a total of 56 mA. This leaves 4 mA sink current capability which can be divided in any way among the remaining 8 I/O lines of the 8243.

NOTE:

A10 to 50 KΩ pullup resistor to +5V should be added to 8243 outputs when driving to 5V CMOS directly.

Example: This example shows how the use of the 20 mA sink capability of Port 7 affects the sinking capability of the other I/O lines.

An 8243 will drive the following loads simultaneously.

2 loads—20 mA @ 1V (Port 7 only)

8 loads—4 mA @ 0.45V

6 loads—3.2 mA @ 0.45V

Is this within the specified limits?

$\epsilon I_{OL} = (2 \times 20) + (8 \times 4) + (6 \times 3.2) = 91.2$ mA.

From the curve: for $I_{OL} = 4$ mA, $\epsilon I_{OL} \cong 93$ mA. Since 91.2 mA < 93 mA the loads are within specified limits.

Although the 20 mA @ 1V loads are used in calculating ϵI_{OL}, it is the largest current required @ 0.45V which determines the maximum allowable ϵI_{OL}.

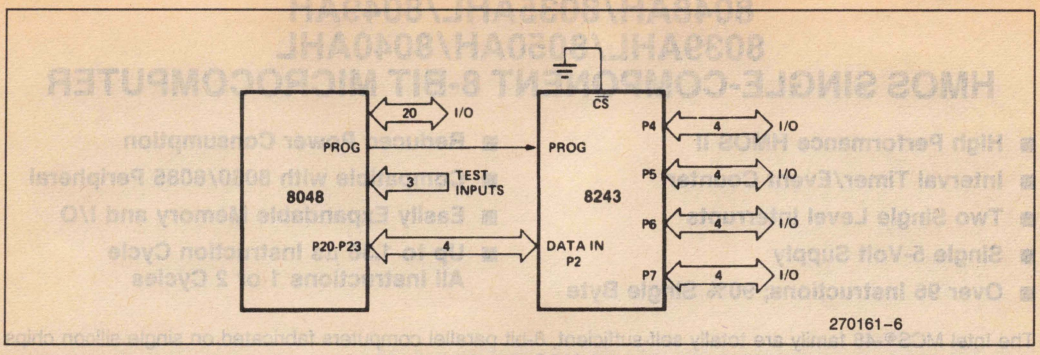


Figure 4. Expander Interface

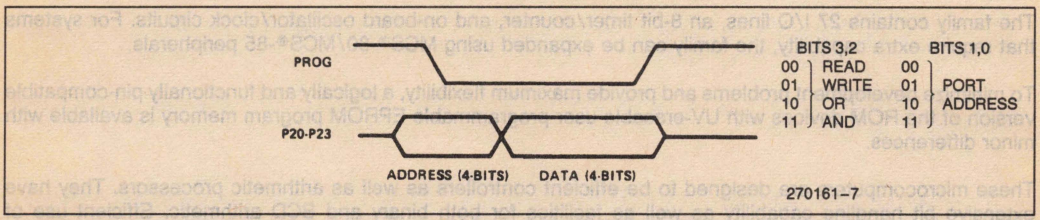


Figure 5. Output Expander Timing

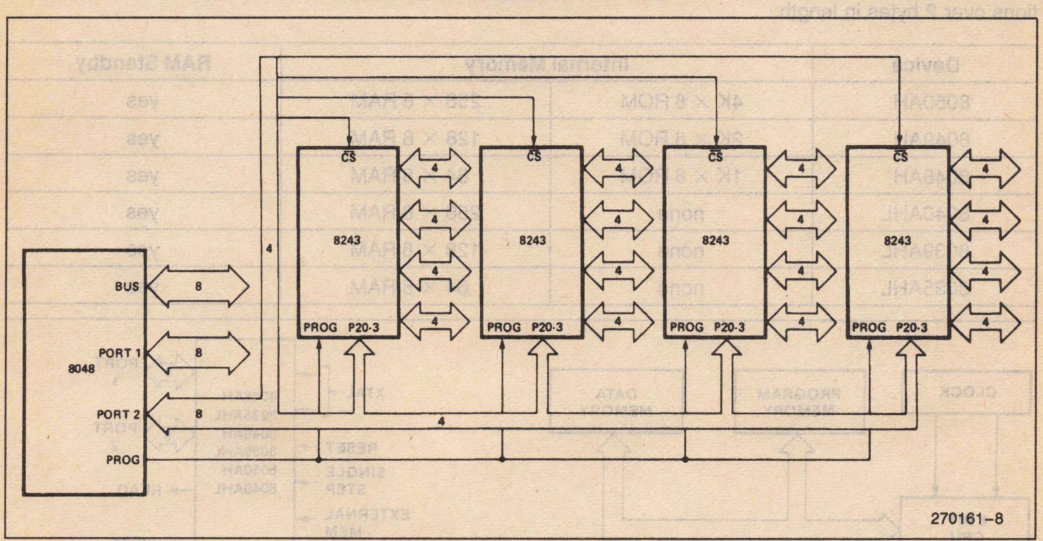


Figure 6. Using Multiple 8243's

8048AH/8035AHL/8049AH
8039AHL/8050AH/8040AHL
HMOS SINGLE-COMPONENT 8-BIT MICROCOMPUTER

- **High Performance HMOS II**
- **Interval Timer/Event Counter**
- **Two Single Level Interrupts**
- **Single 5-Volt Supply**
- **Over 96 Instructions; 90% Single Byte**
- **Reduced Power Consumption**
- **Compatible with 8080/8085 Peripheral**
- **Easily Expandable Memory and I/O**
- **Up to 1.36 μs Instruction Cycle All Instructions 1 or 2 Cycles**

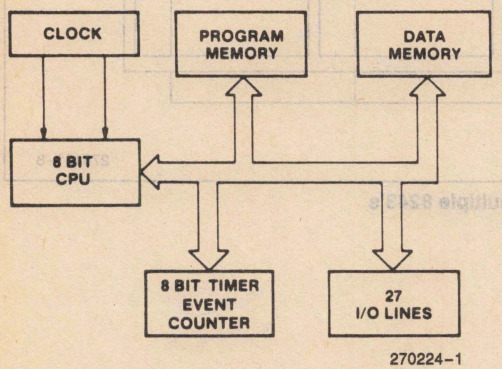
The Intel MCS®-48 family are totally self-sufficient, 8-bit parallel computers fabricated on single silicon chips using Intel's advanced N-channel silicon gate HMOS process.

The family contains 27 I/O lines, an 8-bit timer/counter, and on-board oscillator/clock circuits. For systems that require extra capability, the family can be expanded using MCS®-80/MCS®-85 peripherals.

To minimize development problems and provide maximum flexibility, a logically and functionally pin-compatible version of the ROM devices with UV-erasable user-programmable EPROM program memory is available with minor differences.

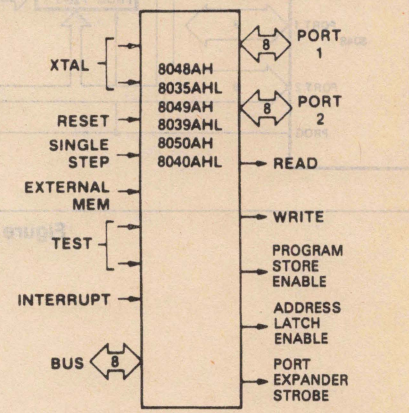
These microcomputers are designed to be efficient controllers as well as arithmetic processors. They have extensive bit handling capability as well as facilities for both binary and BCD arithmetic. Efficient use of program memory results from an instruction set consisting mostly of single byte instructions and no instructions over 2 bytes in length.

Device	Internal Memory		RAM Standby
8050AH	4K × 8 ROM	256 × 8 RAM	yes
8049AH	2K × 8 ROM	128 × 8 RAM	yes
8048AH	1K × 8 ROM	64 × 8 RAM	yes
8040AHL	none	256 × 8 RAM	yes
8039AHL	none	128 × 8 RAM	yes
8035AHL	none	64 × 8 RAM	yes



270224-1

Figure 1. Block Diagram



270224-2

Figure 2. Logic Symbol

December 1986
Order Number: 270224-001

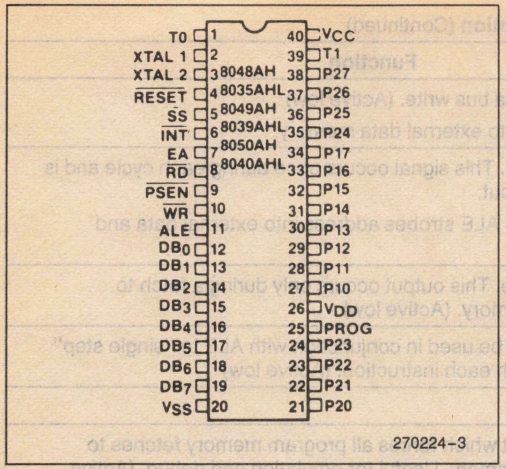


Figure 3. Pin Configuration

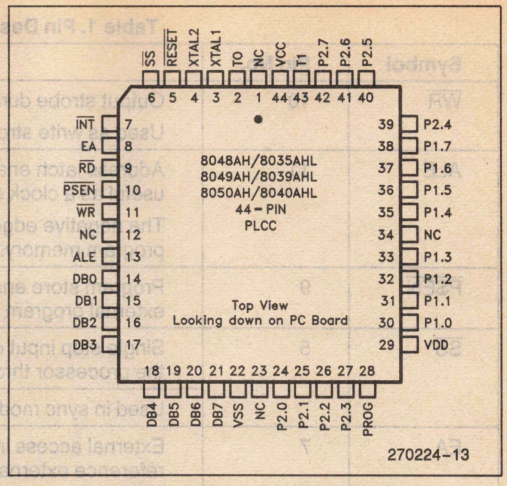


Figure 4. Pad Configuration

Table 1. Pin Description (40 Pin DIP)

Symbol	Pin No.	Function
V_{SS}	20	Circuit GND potential.
V_{DD}	26	+5V during normal operation.
		Low power standby pin.
V_{CC}	40	Main power supply; +5V during operation.
PROG	25	Output strobe for 8243 I/O expander.
P10–P17 Port 1	27–34	8-bit quasi-bidirectional port.
P20–P23 P24–P27 Port 2	21–24 35–38	8-bit quasi-bidirectional port. P20–P23 contain the four high order program counter bits during an external program memory fetch and serve as a 4-bit I/O expander bus for 8243.
DB0–DB7 BUS	12–19	True bidirectional port which can be written or read synchronously using the RD, WR strobes. The port can also be statically latched.
		Contains the 8 low order program counter bits during an external program memory fetch, and receives the addressed instruction under the control of PSEN. Also contains the address and data during an external RAM data store instruction, under control of ALE, RD, and WR.
T0	1	Input pin testable using the conditional transfer instructions JT0 and JNT0. T0 can be designated as a clock output using ENT0 CLK instruction.
T1	39	Input pin testable using the JT1, and JNT1 instructions. Can be designated the timer/counter input using the STRT CNT instruction.
INT	6	Interrupt input. Initiates an interrupt if interrupt is enabled. Interrupt is disabled after a reset. Also testable with conditional jump instruction. (Active Low) interrupt must remain low for at least 3 machine cycles for proper operation.
RD	8	Output strobe activated during a BUS read. Can be used to enable data onto the bus from an external device.
		Used as a road strobe to external data memory. (Active low)
RESET	4	Input which is used to initialize the processor. (Active low) (Non TTL V_{IH})
		Used during power down.
		Used during ROM verification.

Table 1. Pin Description (Continued)

Symbol	Pin No.	Function
$\overline{WR}$	10	Output strobe during a bus write. (Active low) Used as write strobe to external data memory.
ALE	11	Address latch enable. This signal occurs once during each cycle and is useful as a clock output. The negative edge of ALE strobes address into external data and program memory.
$\overline{PSEN}$	9	Program store enable. This output occurs only during a fetch to external program memory. (Active low)
$\overline{SS}$	5	Single step input can be used in conjunction with ALE to "single step" the processor through each instruction. (Active low)
		Used in sync mode.
EA	7	External access input which forces all program memory fetches to reference external memory. Useful for emulation and debug. (Active high)
		Used during ROM verification. (12V)
XTAL1	2	One side of crystal input for internal oscillator. Also input for external source. (Non TTL V_{IH})
XTAL2	3	Other side of crystal input.

Table 2. Instruction Set

Mnemonic	Description	Bytes	Cycles
ACCUMULATOR			
ADD A, R	Add register to A	1	1
ADD A,@R	Add data memory to A	1	1
ADD A, #data	Add immediate to A	2	2
ADDC A, R	Add register with carry	1	1
ADDC A, @R	Add data memory with carry	1	1
ADDC A, #data	Add immediate with carry	2	2
ANL A, R	And register to A	1	1
ANL A, @R	And data memory to A	1	1
ANL A, #data	And immediate to A	2	2
ORL A, R	Or register to A	1	1
ORL A, @R	Or data memory to A	1	1
ORL A, #data	Or immediate to A	2	2
XRL A, R	Exclusive or register to A	1	1
XRL A, @R	Exclusive or data memory to A	1	1
XRL, A, #data	Exclusive or immediate to A	2	2
INC A	Increment A	1	1

Mnemonic	Description	Bytes	Cycles
ACCUMULATOR (Continued)			
DEC A	Decrement A	1	1
CLR A	Clear A	1	1
CPL A	Complement A	1	1
DA A	Decimal adjust A	1	1
SWAP A	Swap nibbles of A	1	1
RL A	Rotate A left	1	1
RLC A	Rotate A left through carry	1	1
RR A	Rotate A right	1	1
RRC A	Rotate A right through carry	1	1
INPUT/OUTPUT			
IN A, P	Input port to A	1	2
OUTL P, A	Output A to port	1	2
ANL P, #data	And immediate to port	2	2
ORL P, #data	Or immediate to port	2	2
INS A, BUS	Input BUS to A	1	2
OUTL BUS, A	Output A to BUS	1	2
ANL BUS, #data	And immediate to BUS	2	2

Table 2. Instruction Set (Continued)

Mnemonic	Description	Bytes	Cycles
INPUT/OUTPUT (Continued)			
ORL BUS, #data	Or immediate to BUS	2	2
MOVD A, P	Input expander port to A	1	2
MOVD P, A	Output A to expander port	1	2
ANLD P, A	And A to expander port	1	2
ORLD P, A	Or A to expander port	1	2
REGISTERS			
INC R	Increment register	1	1
INC @R	Increment data memory	1	1
DEC R	Decrement register	1	1
BRANCH			
JMP addr	Jump unconditional	2	2
JMPP @A	Jump indirect	1	2
DJNZ R, addr	Decrement register and skip	2	2
JC addr	Jump on carry = 1	2	2
JNC addr	Jump on carry = 0	2	2
JZ addr	Jump on A zero	2	2
JNZ addr	Jump on A not zero	2	2
JT0 addr	Jump on T0 = 1	2	2
JNT0 addr	Jump on T0 = 0	2	2
JT1 addr	Jump on T1 = 1	2	2
JNT1 addr	Jump on T1 = 0	2	2
JF0 addr	Jump on F0 = 1	2	2
JF1 addr	Jump on F1 = 1	2	2
JTF addr	Jump on timer flag	2	2
JNI addr	Jump on INT = 0	2	2
JBb addr	Jump on accumulator bit	2	2
SUBROUTINE			
CALL addr	Jump to subroutine	2	2
RET	Return	1	2
RETR	Return and restore status	1	2
FLAGS			
CLR C	Clear carry	1	1
CPL C	Complement carry	1	1
CLR F0	Clear flag 0	1	1
CPL F0	Complement flag 0	1	1
CLR F1	Clear flag 1	1	1
CPL F1	Complement flag 1	1	1
DATA MOVES			
MOV A, R	Move register to A	1	1
MOVE A, @R	Move data memory to A	1	1

Mnemonic	Description	Bytes	Cycles
DATA MOVES (Continued)			
Move A, #data	Move immediate to A	2	2
MOV R, A	Move A to register	1	1
MOV @R, A	Move A to data memory	1	1
MOV R, #data	Move immediate to register	2	2
MOV @R, #data	Move immediate to data memory	2	2
MOV A, PSW	Move PSW to A	1	1
MOV PSW, A	Move A to PSW	1	1
XCH A, R	Exchange A and register	1	1
XCH A, @R	Exchange A and data memory	1	1
XCHD A, @R	Exchange nibble of A and register	1	1
MOVX A, @R	Move A to external data memory to A	1	2
MOVX @R, A	Move A external data data memory	1	2
MOVP A, @A	Move to A from current page	1	2
MOVP3 A, @	Move to A from page 3	1	2
TIMER/COUNTER			
MOV A, T	Read timer/counter	1	1
MOV T, A	Load timer/counter	1	1
STRT T	Start timer	1	1
STRT CNT	Start timer	1	1
STOP TCNT	Stop timer/counter	1	1
EN TCNTI	Enable timer/counter interrupt	1	1
DIS TCNTI	Disable timer/counter interrupt	1	1
CONTROL			
EN I	Enable external interrupt	1	1
DIS I	Disable external interrupt	1	1
SEL RB0	Select register bank 0	1	1
SEL RB1	Select register bank 1	1	1
SEL MB0	Select memory bank 0	1	1
SEL MB1	Select memory bank 1	1	1
ENT0 CLK	Enable clock output on T0	1	1
NOP	No operation	1	1
IDL	Select Idle Operation	1	1

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias . . . . . 0°C to 70°C
Storage Temperature . . . . . −65°C to +150°C
Voltage on any Pin with Respect to Ground . . . . . −0.5V to +7V
Power Dissipation 1.5W

Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

NOTICE: Specifications contained within the following tables are subject to change.

D.C. CHARACTERISTICS T_A = 0°C to 70°C; V_{CC} = V_{DD} = 5V ±10%; V_{SS} = 0V

Symbol	Parameter	Limits			Unit	Test Conditions	Device
		Min	Typ	Max			
V_{IL}	Input Low Voltage (All Except RESET, X1, X2)	−0.5		0.8	V		All
V_{IL1}	Input Low Voltage (RESET, X1, X2)	−5		0.6	V		All
V_{IH}	Input High Voltage (All Except XTAL1, XTAL2, RESET)	2.0		V_{CC}	V		All
V_{IH1}	Input High Voltage (X1, X2, RESET)	3.8		V_{CC}	V		All
V_{OL}	Output Low Voltage (BUS)			0.45	V	I_{OL} = 2.0 mA	All
V_{OL1}	Output Low Voltage (RD, WR, PSEN, ALE)			0.45	V	I_{OL} = 1.8 mA	All
V_{OL2}	Output Low Voltage (PROG)			0.45	V	I_{OL} = 1.0 mA	All
V_{OL3}	Output Low Voltage (All Other Outputs)			0.45	V	I_{OL} = 1.6 mA	All
V_{OH}	Output High Voltage (BUS)	2.4			V	I_{OH} = −400 μA	All
V_{OH1}	Output High Voltage (RD, WR, PSEN, ALE)	2.4			V	I_{OH} = −100 μA	All
V_{OH2}	Output High Voltage (All Other Outputs)	2.4			V	I_{OH} = −40 μA	All

D.C. CHARACTERISTICS T_A = 0°C to 70°C; V_{CC} = V_{DD} = 5V ±10%; V_{SS} = 0V (Continued)

Symbol	Parameter	Limits			Unit	Test Conditions	Device
		Min	Typ	Max			
I_{L1}	Leakage Current (T1, $\overline{INT}$)			±10	µA	$V_{SS} \le V_{IN} \le V_{CC}$	All
I_{LI1}	Input Leakage Current (P10–P17, P20–P27, EA, $\overline{SS}$)			−500	µA	$V_{SS} + 0.45 \le V_{IN} \le V_{CC}$	All
I_{LI2}	Input Leakage Current $\overline{RESET}$	−20		−300	µA	$V_{SS} \le V_{IN} \le 3.8V$	All
I_{L0}	Leakage Current (BUS, T0) (High Impedance State)			±10	µA	$V_{SS} \le V_{IN} \le V_{CC}$	All
I_{DD}	V_{DD} Supply Current (RAM Standby)		3	5	mA		8048AH 8035AHL
			4	7	mA		8049AH 8039AHL
			5	10	mA		8050AH 8040AHL
I_{DD} + I_{CC}	Total Supply Current*		30	65	mA		8048AH 8035AHL
			35	70	mA		8049AH 8039AHL
			40	80	mA		8050AH 8040AHL
V_{DD}	RAM Standby Voltage	2.2		5.5	V	Standby Mode Reset ≤ V_{IL1}	All

NOTE:
*I_{CC} + I_{DD} is measured with all output disconnected; $\overline{SS}$, $\overline{RESET}$, and $\overline{INT}$ equal to V_{CC}; EA equal to V_{SS}.

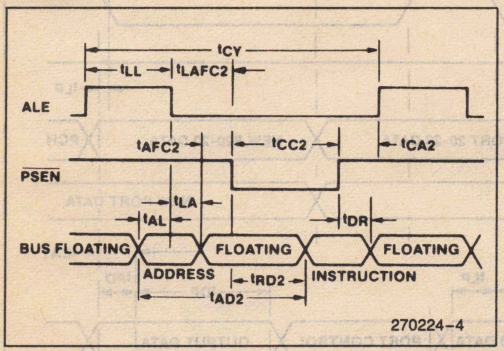
A.C. CHARACTERISTICS $T_A = 0°C$ to $70°C$; $V_{CC} = V_{DD} = 5V \pm 10\%$; $V_{SS} = 0V$

Symbol	Parameter	f(t) (Note 3)	11 MHz		Unit	Conditions (Note 1)
			Min	Max		
t	Clock Period	1/xtal freq	90.9	1000	ns	(Note 3)
t_{LL}	ALE Pulse Width	3.5t − 170	150		ns	
t_{AL}	Addr Setup to ALE	2t − 110	70		ns	(Note 2)
t_{LA}	Addr Hold from ALE	t − 40	50		ns	
t_{CC1}	Control Pulse Width ($\overline{RD}$, $\overline{WR}$)	7.5t − 200	480		ns	
t_{CC2}	Control Pulse Width ($\overline{PSEN}$)	6t − 200	350		ns	
t_{DW}	Data Setup before $\overline{WR}$	6.5t − 200	390		ns	
t_{WD}	Data Hold after $\overline{WR}$	t − 50	40		ns	
t_{DR}	Data Hold ($\overline{RD}$, $\overline{PSEN}$)	1.5t − 30	0	110	ns	
t_{RD1}	$\overline{RD}$ to Data in	6t − 170		375	ns	
t_{RD2}	$\overline{PSEN}$ to Data in	4.5t − 170		240	ns	
t_{AW}	Addr Setup to $\overline{WR}$	5t − 150	300		ns	
t_{AD1}	Addr Setup to Data ($\overline{RD}$)	10.5t − 220		730	ns	
t_{AD2}	Addr Setup to Data ($\overline{PSEN}$)	7.5t − 200		460	ns	
t_{AFC1}	Addr Float to $\overline{RD}$, $\overline{WR}$	2t − 40	140		ns	(Note 2)
t_{AFC2}	Addr Float to $\overline{PSEN}$	0.5t − 40	10		ns	(Note 2)
t_{LAFC1}	ALE to Control ($\overline{RD}$, $\overline{WR}$)	3t − 75	200		ns	
t_{LAFC2}	ALE to Control ($\overline{PSEN}$)	1.5t − 75	60		ns	
t_{CA1}	Control to ALE ($\overline{RD}$, $\overline{WR}$, $\overline{PROG}$)	t − 65	25		ns	
t_{CA2}	Control to ALE ($\overline{PSEN}$)	4t − 70	290		ns	
t_{CP}	Port Control Setup to $\overline{PROG}$	1.5t − 80	50		ns	
t_{PC}	Port Control Hold to $\overline{PROG}$	4t − 260	100		ns	
t_{PR}	$\overline{PROG}$ to P2 Input Valid	8.5t − 120		650	ns	
t_{PF}	Input Data Hold from $\overline{PROG}$	1.5t	0	140	ns	
t_{DP}	Output Data Setup	6t − 290	250		ns	
t_{PD}	Output Data Hold	1.5t − 90	40		ns	
t_{PP}	$\overline{PROG}$ Pulse Width	10.5t − 250	700		ns	
t_{PL}	Port 2 I/O Setup to ALE	4t − 200	160		ns	
t_{LP}	Port 2 I/O Hold to ALE	0.5t − 30	15		ns	
t_{PV}	Port Output from ALE	4.5t+100		510	ns	
t_{0PRR}	T0 Rep Rate	3t	270		ns	
t_{CY}	Cycle Time	15t	1.36	15.0	μs	

NOTES:
1. Control Outputs CL = 80 pF, BUS Outputs CL = 150 pF.
2. BUS High Impedance Load 20 pF.
3. f(t) assumes 50% duty cycle on X1, X2. Max clock period is for a 1 MHz crystal input.

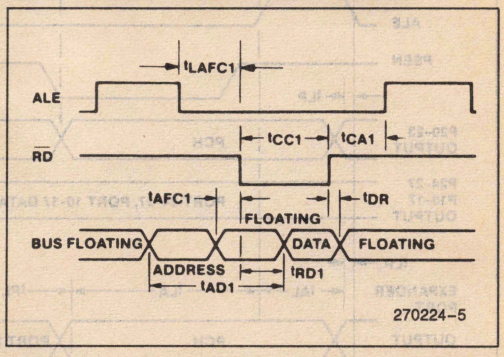
WAVEFORMS

INSTRUCTION FETCH FROM PROGRAM MEMORY



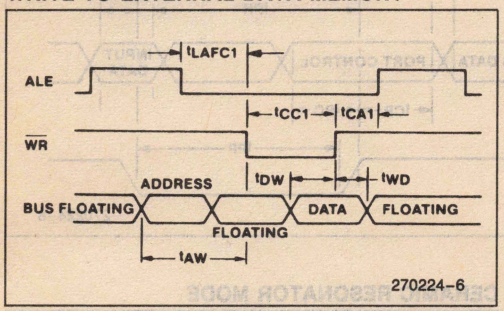
270224-4

READ FROM EXTERNAL DATA MEMORY



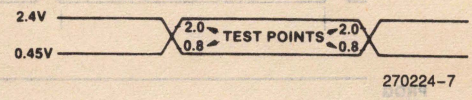
270224-5

WRITE TO EXTERNAL DATA MEMORY



270224-6

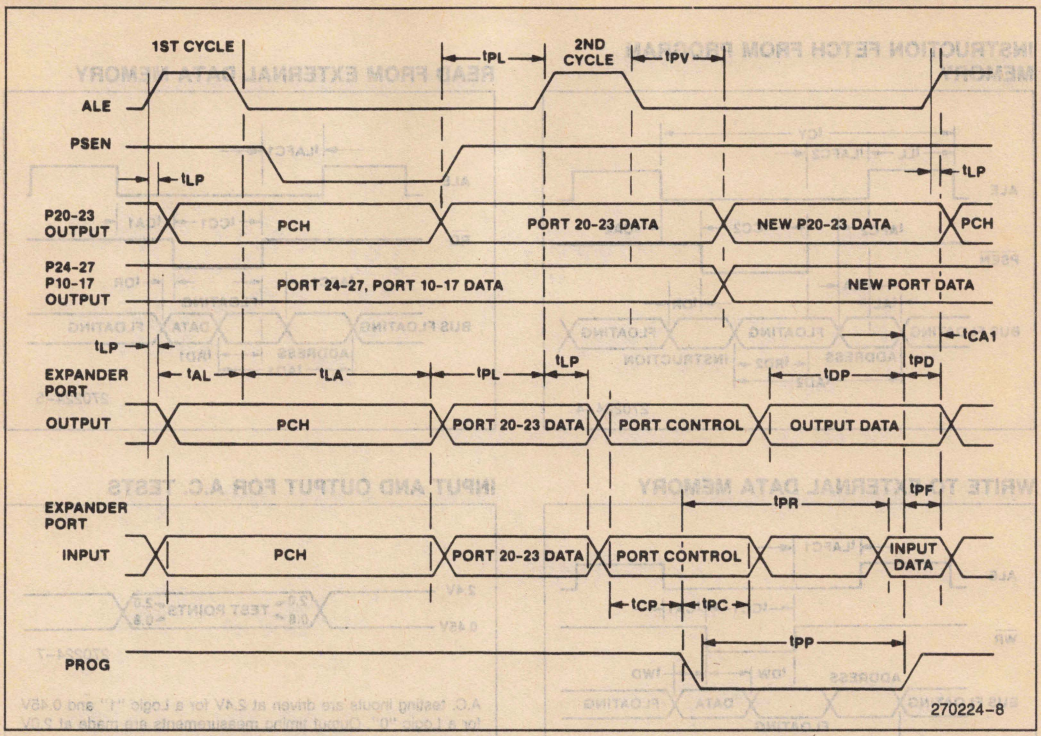
INPUT AND OUTPUT FOR A.C. TESTS



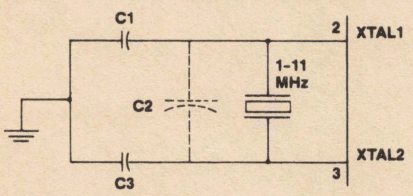
270224-7

A.C. testing inputs are driven at 2.4V for a Logic "1" and 0.45V for a Logic "0". Output timing measurements are made at 2.0V for a Logic "1" and 0.8V for a Logic "0".

PORT 1/PORT 2 TIMING



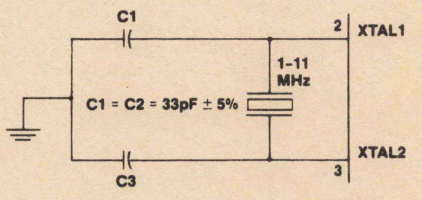
CRYSTAL OSCILLATOR MODE



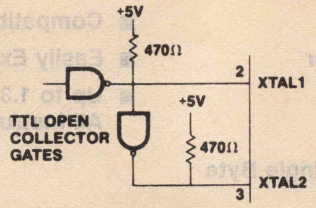
C1 = 5 pF ± ½ pF + (STRAY < 5 pF)
C2 = (CRYSTAL + STRAY) < 8 pF
C3 = 20 pF ± 1 pF (STRAY < 5 pF)

Crystal series resistance should be less than 30Ω at 11 MHz; less than 75Ω at 6 MHz; less than 180Ω at 3.6 MHz.

CERAMIC RESONATOR MODE



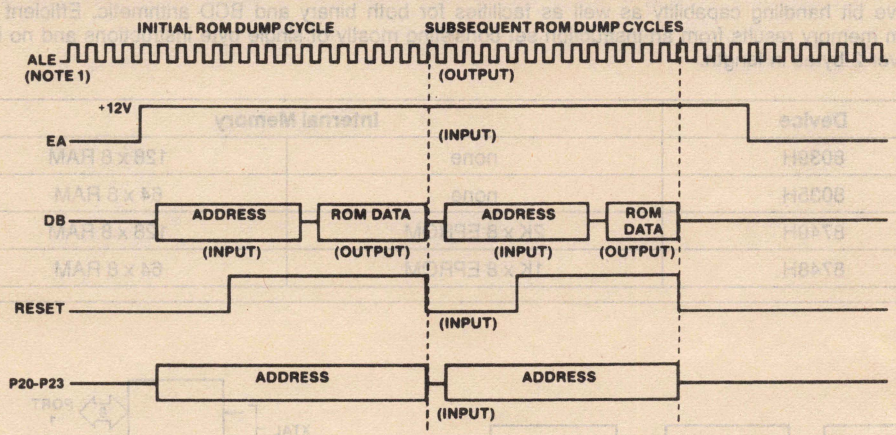
DRIVING FROM EXTERNAL SOURCE



270224-11

For XTAL1 and XTAL2 define "high" as voltages above 1.6V and "low" as voltages below 1.6V. The duty cycle requirements for externally driving XTAL1 and XTAL2 using the circuit shown above are as follows: XTAL1 must be high 35–65% of the period and XTAL2 must be high 36–65% of the period. Rise and fall times must be faster than 20 ns.

SUGGESTED ROM VERIFICATION ALGORITHM FOR H-MOS DEVICE ONLY



270224-12

	48H	49H	50H
A10	0	ADDR	ADDR
A11	0	0	ADDR

$V_{CC} = V_{DD} = +5V$
$V_{SS} = 0V$

NOTE:
ALE is function of X1, X2 inputs.

8748H/8035H/8749H/8039H
HMOS-E SINGLE-COMPONENT 8-BIT MICROCOMPUTER

- **High Performance HMOS-E**
- **Interval Timer/Event Counter**
- **Two Single Level Interrupts**
- **Single 5-Volt Supply**
- **Over 96 Instructions; 90% Single Byte**
- **Compatible with 8080/8085 Peripherals**
- **Easily Expandable Memory and I/O**
- **Up to 1.35 μs Instruction Cycle; All Instructions 1 or 2 Cycles**

The Intel 8749H/8039H/8748H/8035H are totally self-sufficient, 8-bit parallel computers fabricated on single silicon chips using Intel's advanced N-channel silicon gate HMOS-E process.

The family contains 27 I/O lines, an 8-bit timer/counter, on-chip RAM and on-board oscillator/clock circuits. For systems that require extra capability, the family can be expanded using MCS®-80/MCS®-85 peripherals.

These microcomputers are designed to be efficient controllers as well as arithmetic processors. They have extensive bit handling capability as well as facilities for both binary and BCD arithmetic. Efficient use of program memory results from an instruction set consisting mostly of single byte instructions and no instructions over 2 bytes in length.

Device	Internal Memory	
8039H	none	128 x 8 RAM
8035H	none	64 x 8 RAM
8749H	2K x 8 EPROM	128 x 8 RAM
8748H	1K x 8 EPROM	64 x 8 RAM

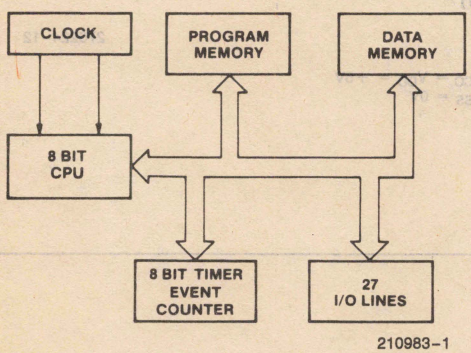


Figure 1. Block Diagram

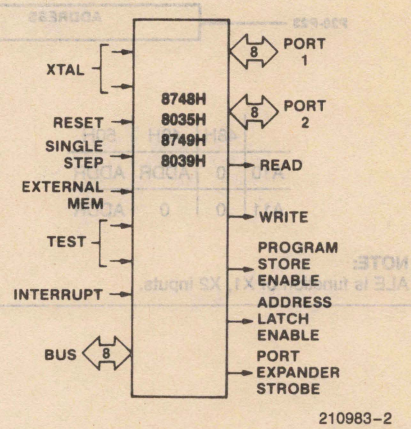


Figure 2. Logic Symbol

November 1986
Order Number: 210983-002

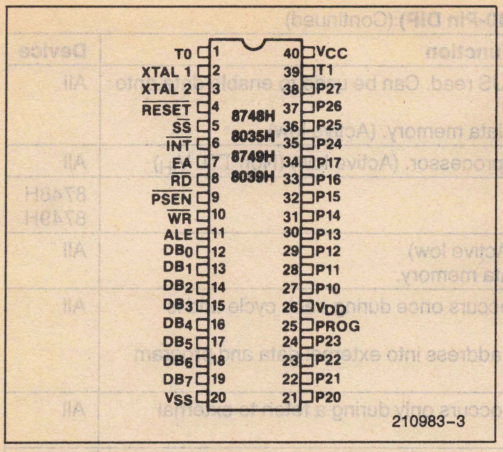


Figure 3. Pin Configuration

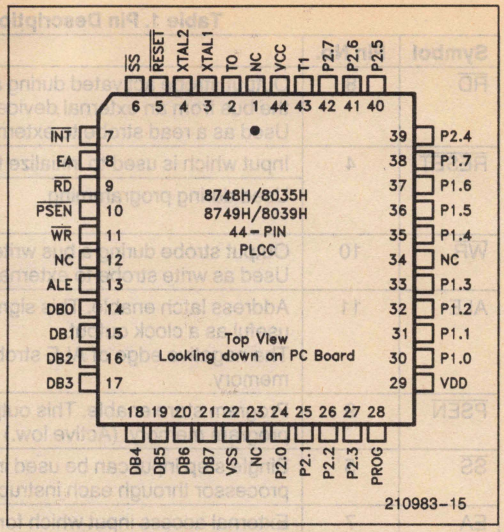


Figure 4. Pad Configuration

Table 1. Pin Description (40-Pin DIP)

Symbol	Pin No.	Function	Device
V_{SS}	20	Circuit GND potential.	All
V_{DD}	26	+5V during normal operation.	All
		Programming power supply (+21V).	8748H 8749H
V_{CC}	40	Main power supply; +5V during operation and programming.	All
PROG	25	Output strobe for 8243 I/O expander.	All
		Program pulse (+18V) input pin during programming.	8748H 8749H (See Note)
P10–P17 Port 1	27–34	8-bit quasi-bidirectional port.	All
P20–P23 P24–P27 Port 2	21–24 35–38	8-bit quasi-bidirectional port. P20–P23 contain the four high order program counter bits during an external program memory fetch and serve as a 4-bit I/O expander bus for 8243.	All
DB0–DB7 BUS	12–19	True bidirectional port which can be written or read synchronously using the $\overline{RD}$, $\overline{WR}$ strobes. The port can also be statically latched. Contains the 8 low order program counter bits during an external program memory fetch, and receives the addressed instruction under the control of $\overline{PSEN}$. Also contains the address and data during an external RAM data store instruction, under control of ALE, $\overline{RD}$, and $\overline{WR}$.	All
T0	1	Input pin testable using the conditional transfer instructions JT0 and JNT0. T0 can be designated as a clock output using ENT0 CKL instruction.	All
		Used during programming.	8748H 8749H
T1	39	Input pin testable using the JT1, and JNT1 instructions. Can be designated the timer/counter input using the STRT CNT instruction.	All
$\overline{INT}$	6	Interrupt input. Initiates an interrupt if interrupt is enabled. Interrupt is disabled after a reset. Also testable with conditional jump instruction. (Active low) interrupt must remain low for at least 3 machine cycles for proper operation.	All

Table 1. Pin Description (40-Pin DIP) (Continued)

Symbol	Pin No.	Function	Device
$\overline{RD}$	8	Output strobe activated during a BUS read. Can be used to enable data onto the bus from an external device. Used as a read strobe to external data memory. (Active low)	All
$\overline{RESET}$	4	Input which is used to initialize the processor. (Active low) (Non TTL V_{IH})	All
		Used during programming.	8748H 8749H
$\overline{WR}$	10	Output strobe during a bus write. (Active low) Used as write strobe to external data memory.	All
ALE	11	Address latch enable. This signal occurs once during each cycle and is useful as a clock output. The negative edge of ALE strobes address into external data and program memory.	All
$\overline{PSEN}$	9	Program store enable. This output occurs only during a fetch to external program memory. (Active low.)	All
$\overline{SS}$	5	Single step input can be used in conjunction with ALE to "single step" the processor through each instruction.	All
EA	7	External access input which forces all program memory fetches to reference external memory. Useful for emulation and debug. (Active high.)	All
		Used during (18V) programming.	8748H 8749H
XTAL1	2	One side of crystal input for internal oscillator. Also input for external source. (Non TTL V_{IH}.)	All
XTAL2	3	Other side of crystal input.	All

NOTE:
On the 8749H/8039H, PROG must be clamped to V_{CC} when not programming. A diode should be used when using an 8243; otherwise, a direct connection is permissible.

Table 2. Instruction Set

Mnemonic	Description	Bytes	Cycles
ACCUMULATOR			
ADD A, R	Add register to A	1	1
ADD A,@R	Add data memory to A	1	1
ADD A, #data	Add immediate to A	2	2
ADDC A, R	Add register with carry	1	1
ADDC A, @R	Add data memory with carry	1	1
ADDC A, #data	Add immediate with carry	2	2
ANL A, R	And register to A	1	1
ANL A, @R	And data memory to A	1	1
ANL A, #data	And immediate to A	2	2
ORL A, R	Or register to A	1	1
ORL A, @R	Or data memory to A	1	1
ORL A, #data	Or immediate to A	2	2
XRL A, R	Exclusive or register to A	1	1
XRL A, @R	Exclusive or data memory to A	1	1
XRL A, #data	Exclusive or immediate to A	2	2
INC A	Increment A	1	1
DEC A	Decrement A	1	1
CLR A	Clear A	1	1
CPL A	Complement A	1	1
DA A	Decimal adjust A	1	1
SWAP A	Swap nibbles of A	1	1
RL A	Rotate A left	1	1
RLC A	Rotate A left through carry	1	1
RR A	Rotate A right	1	1
RRC A	Rotate A right through carry	1	1
INPUT/OUTPUT			
IN A, P	Input port to A	1	2
OUTL P, A	Output A to port	1	2
ANL P, #data	And immediate to port	2	2

Table 2. Instruction Set (Continued)

Mnemonic	Description	Bytes	Cycles
INPUT/OUTPUT (Continued)			
ORL P, #data	Or immediate to port	2	2
INS A, BUS	Input BUS to A	1	2
OUTL BUS, A	Output A to BUS	1	2
ANL BUS, #data	And immediate to BUS	2	2
ORL BUS, #data	Or immediate to BUS	2	2
MOVD A, P	Input expander port to A	1	2
MOVD P, A	Output A to expander port	1	2
ANLD P, A	And A to expander port	1	2
ORLD P, A	Or A to expander port	1	2
REGISTERS			
INC R	Increment register	1	1
INC @R	Increment data memory	1	1
DEC R	Decrement register	1	1
BRANCH			
JMP addr	Jump unconditional	2	2
JMPP @A	Jump indirect	1	2
DJNZ R, addr	Decrement register and skip	2	2
JC addr	Jump on carry = 1	2	2
JNC addr	Jump on carry = 0	2	2
JZ addr	Jump on A zero	2	2
JNZ addr	Jump on A not zero	2	2
JT0 addr	Jump on T0 = 1	2	2
JNT0 addr	Jump on T0 = 0	2	2
JT1 addr	Jump on T1 = 1	2	2
JNT1 addr	Jump on T1 = 0	2	2
JF0 addr	Jump on F0 = 1	2	2
JF1 addr	Jump on F1 = 1	2	2
JTF addr	Jump on timer flag	2	2
JNI addr	Jump on INT = 0	2	2
JBb addr	Jump on accumulator bit	2	2
SUBROUTINE			
CALL addr	Jump to subroutine	2	2
RET	Return	1	2
RETR	Return and restore status	1	2
FLAGS			
CLR C	Clear carry	1	1
CPL C	Complement carry	1	1
CLR F0	Clear flag 0	1	1
CPL F0	Complement flag 0	1	1
CLR F1	Clear flag 1	1	1
CPL F1	Complement flag 1	1	1

Mnemonic	Description	Bytes	Cycles
DATA MOVES			
MOV A, R	Move register to A	1	1
MOV A, @R	Move data memory to A	1	1
MOV A, #data	Move immediate to A	2	2
MOV R, A	Move A to register	1	1
MOV @R, A	Move A to data memory	1	1
MOV R, #data	Move immediate to register	2	2
MOV @R, #data	Move immediate to data memory	2	2
MOV A, PSW	Move PSW to A	1	1
MOV PSW, A	Move A to PSW	1	1
XCH A, R	Exchange A and register	1	1
XCH A, @R	Exchange A and data memory	1	1
XCHD A, @R	Exchange nibble of A and register	1	1
MOVX A, @R	Move external data memory to A	1	2
MOVX @R, A	Move A to external data memory	1	2
MOVP A, @A	Move to A from current page	1	2
MOVP3 A, @A	Move to A from page 3	1	2
TIMER/COUNTER			
MOV A, T	Read timer/counter	1	1
MOV T, A	Load timer/counter	1	1
STRT T	Start timer	1	1
STRT CNT	Start counter	1	1
STOP TCNT	Stop timer/counter	1	1
EN TCNTI	Enable timer/counter interrupt	1	1
DIS TCNTI	Disable timer/counter interrupt	1	1
CONTROL			
EN I	Enable external interrupt	1	1
DIS I	Disable external interrupt	1	1
SEL RB0	Select register bank 0	1	1
SEL RB1	Select register bank 1	1	1
SEL MB0	Select memory bank 0	1	1
SEL MB1	Select memory bank 1	1	1
ENT0 CLK	Enable clock output on T0	1	1
NOP	No operation	1	1

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias .....0°C to 70°C
Storage Temperature .....−65°C to +150°C
Voltage On Any Pin With Respect to Ground.....−0.5V to +7V
Power Dissipation.....1.0 Watt

Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

NOTICE: Specifications contained within the following tables are subject to change.

D.C. CHARACTERISTICS T_A = 0°C to 70°C; V_{CC} = V_{DD} = 5V ±10%; V_{SS} = 0V

Symbol	Parameter	Limits			Unit	Test Conditions	Device
		Min	Typ	Max			
V_{IL}	Input Low Voltage (All Except RESET, X1, X2)	−0.5		0.8	V		All
V_{IL1}	Input Low Voltage (RESET, X1, X2)	−0.5		0.6	V		All
V_{IH}	Input High Voltage (All Except XTAL1, XTAL2, RESET	2.0		V_{CC}	V		All
V_{IH1}	Input High Voltage (X1, X2, RESET)	3.8		V_{CC}	V		All
V_{OL}	Output Low Voltage (BUS)			0.45	V	I_{OL} = 2.0 mA	All
V_{OL1}	Output Low Voltage (RD, WR, PSEN, ALE)			0.45	V	I_{OL} = 1.8 mA	All
V_{OL2}	Output Low Voltage (PROG)			0.45	V	I_{OL} = 1.0 mA	All
V_{OL3}	Output Low Voltage (All Other Outputs)			0.45	V	I_{OL} = 1.6 mA	All
V_{OH}	Output High Voltage (BUS)	2.4			V	I_{OH} = −400 μA	All
V_{OH1}	Output High Voltage (RD, WR, PSEN, ALE)	2.4			V	I_{OH} = −100 μA	All
V_{OH2}	Output High Voltage (All Other Outputs)	2.4			V	I_{OH} = −40 μA	All
I_{L1}	Leakage Current (T1, INT)			±10	μA	$V_{SS} \le V_{IN} \le V_{CC}$	All
I_{LI1}	Input Leakage Current (P10–P17, P20–P27, EA, SS)			−500	μA	V_{SS} + 0.45 ≤ $V_{IN} \le V_{CC}$	All
I_{LI2}	Input Leakage Current RESET	−10		−300	μA	$V_{SS} \le V_{IN} \le$ 3.8V	All
I_{LO}	Leakage Current (BUS, T0) (High Impedance State)			±10	μA	$V_{SS} \le V_{IN} \le V_{CC}$	All
$I_{DD} + I_{CC}$	Total Supply Current*		80	100	mA		8035H
			95	110	mA		8039H
			80	100	mA		8048H
			95	110	mA		8749H

NOTE:
*$I_{CC} + I_{DD}$ is measured with all outputs disconnected; SS, RESET, and INT equal to V_{CC}; EA equal to V_{SS}.

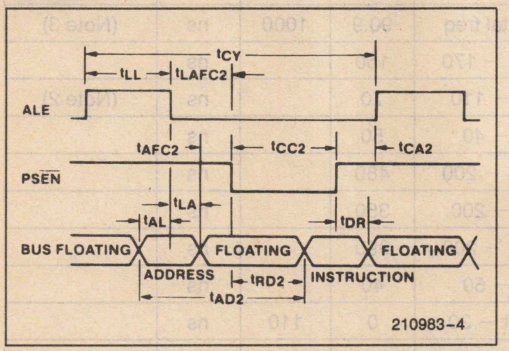
A.C. CHARACTERISTICS T_A = 0°C to 70°C; V_{CC} = V_{DD} = 5V ±10%; V_{SS} = 0V

Symbol	Parameter	f(t) (Note 3)	11 MHz		Unit	Conditions (Note 1)
			Min	Max		
t	Clock Period	1/xtal freq	90.9	1000	ns	(Note 3)
t_{LL}	ALE Pulse Width	3.5t − 170	150		ns	
t_{AL}	Addr Setup to ALE	2t − 110	70		ns	(Note 2)
t_{LA}	Addr Hold from ALE	t − 40	50		ns	
t_{CC1}	Control Pulse Width ($\overline{RD}$, $\overline{WR}$)	7.5t − 200	480		ns	
t_{CC2}	Control Pulse Width ($\overline{PSEN}$)	6t − 200	350		ns	
t_{DW}	Data Setup before $\overline{WR}$	6.5t − 200	390		ns	
t_{WD}	Data Hold after $\overline{WR}$	t − 50	40		ns	
t_{DR}	Data Hold ($\overline{RD}$, $\overline{PSEN}$)	1.5t − 30	0	110	ns	
t_{RD1}	$\overline{RD}$ to Data In	6t − 170		375	ns	
t_{RD2}	$\overline{PSEN}$ to Data In	4.5t − 170		240	ns	
t_{AW}	Addr Setup to $\overline{WR}$	5t − 150	300		ns	
t_{AD1}	Addr Setup to Data ($\overline{RD}$)	10.5t − 220		730	ns	
t_{AD2}	Addr Setup to Data ($\overline{PSEN}$)	7.5t − 200		460	ns	
t_{AFC1}	Addr Float to $\overline{RD}$, $\overline{WR}$	2t − 40	140		ns	(Note 2)
t_{AFC2}	Addr Float to $\overline{PSEN}$	0.5t − 40	10		ns	(Note 2)
t_{LAFC1}	ALE to Control ($\overline{RD}$, $\overline{WR}$)	3t − 75	200		ns	
t_{LAFC2}	ALE to Control ($\overline{PSEN}$)	1.5t − 75	60		ns	
t_{CA1}	Control to ALE ($\overline{RD}$, $\overline{WR}$, $\overline{PROG}$)	t − 65	25		ns	
t_{CA2}	Control to ALE ($\overline{PSEN}$)	4t − 70	290		ns	
t_{CP}	Port Control Setup to $\overline{PROG}$	1.5t − 80	50		ns	
t_{PC}	Port Control Hold to $\overline{PROG}$	4t − 260	100		ns	
t_{PR}	$\overline{PROG}$ to P2 Input Valid	8.5t − 120		650	ns	
t_{PF}	Input Data Hold from $\overline{PROG}$	1.5t	0	140	ns	
t_{DP}	Output Data Setup	6t − 290	250		ns	
t_{PD}	Output Data Hold	1.5t − 90	40		ns	
t_{PP}	$\overline{PROG}$ Pulse Width	10.5t − 250	700		ns	
t_{PL}	Port 2 I/O Setup to ALE	4t − 200	160		ns	
t_{LP}	Port 2 I/O Hold to ALE	0.5t − 30	15		ns	
t_{PV}	Port Output from ALE	4.5t + 100		510	ns	
t_{0PRR}	T0 Rep Rate	3t	270		ns	
t_{CY}	Cycle Time	15t	1.36	15.0	μs	

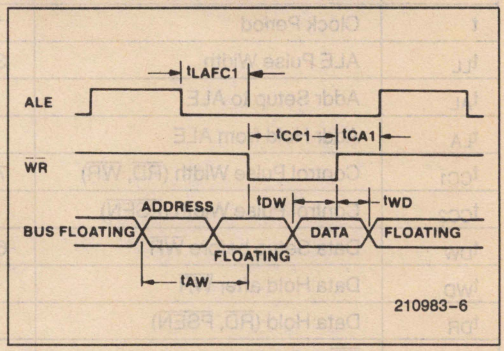
NOTES:
1. Control outputs CL = 80 pF; BUS outputs CL = 150 pF.
2. BUS High Impedance Load 20 pF.
3. f(t) assumes 50% duty cycle on X1, X2. Max clock period is for a 1 MHz crystal input.

WAVEFORMS

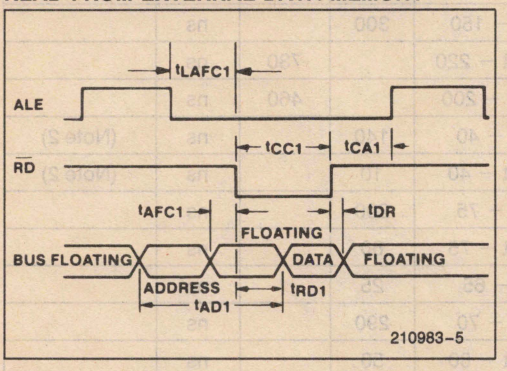
INSTRUCTION FETCH FROM PROGRAM MEMORY



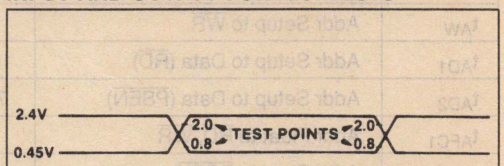
WRITE TO EXTERNAL DATA MEMORY



READ FROM EXTERNAL DATA MEMORY

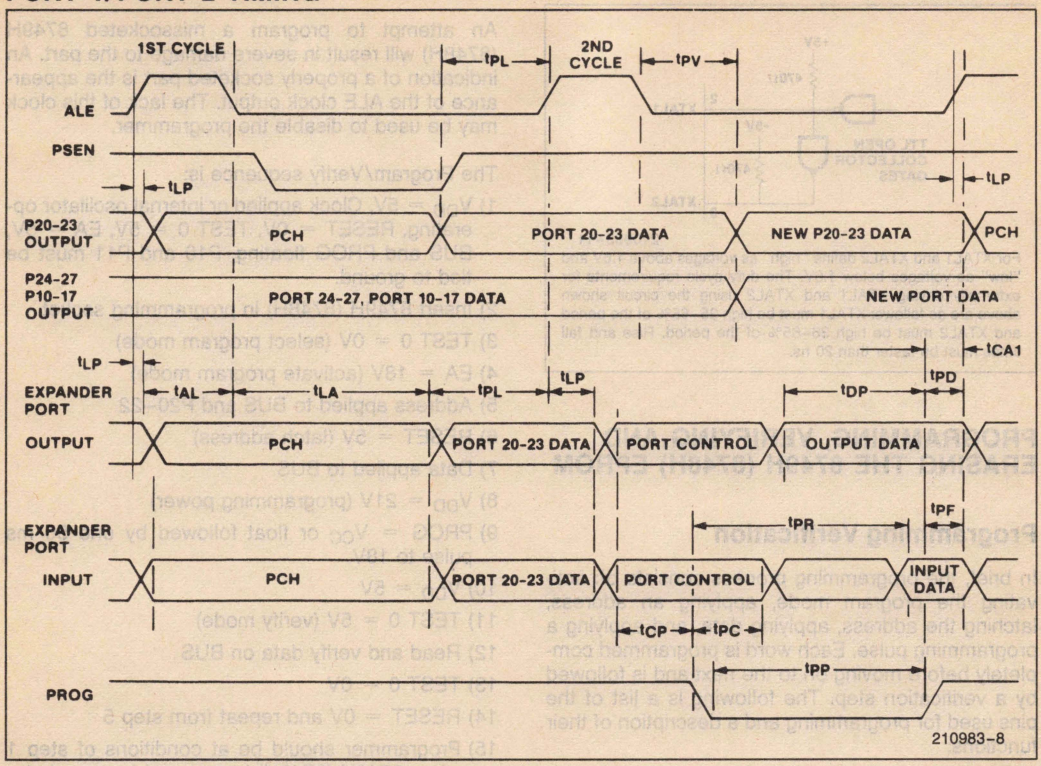


INPUT AND OUTPUT FOR A.C. TESTS

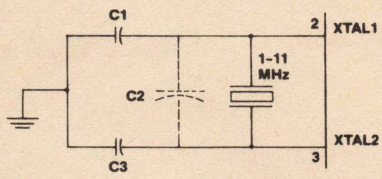


A.C. testing inputs are driven at 2.4V for a Logic "1" and 0.45V for a Logic "0". Output timing measurements are made at 2.0V for a Logic "1" and 0.8V for a Logic "0."

PORT 1/PORT 2 TIMING



CRYSTAL OSCILLATOR MODE

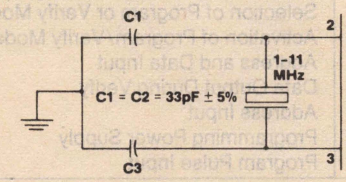


210983-9

C1 = 5 pF ±½ pF + (STRAY < 5 pF)
C2 = (CRYSTAL + STRAY) < 8 pF
C3 = 20 pF ±1 pF (STRAY < 5 pF)

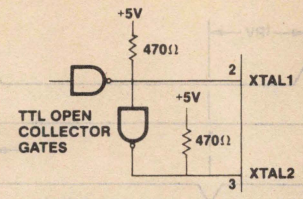
Crystal series resistance should be less than 30Ω at 11 MHz; less than 75Ω at 6 MHz; less than 180Ω at 3.6 MHz.

CERAMIC RESONATOR MODE



210983-10

DRIVING FROM EXTERNAL SOURCE



210983-11

For XTAL1 and XTAL2 define "high" as voltages above 1.6V and "low" as voltages below 1.6V. The duty cycle requirements for externally driving XTAL1 and XTAL2 using the circuit shown above are as follows: XTAL1 must be high 35–65% of the period and XTAL2 must be high 36–65% of the period. Rise and fall times must be faster than 20 ns.

PROGRAMMING, VERIFYING AND ERASING THE 8749H (8748H) EPROM

Programming Verification

In brief, the programming process consists of: activating the program mode, applying an address, latching the address, applying data, and applying a programming pulse. Each word is programmed completely before moving on to the next and is followed by a verification step. The following is a list of the pins used for programming and a description of their functions:

Pin	Function
XTAL 1	Clock Input (3 to 4.0 MHz)
XTAL 2	
RESET	Initialization and Address Latching
TEST 0	Selection of Program or Verify Mode
EA	Activation of Program/Verify Modes
BUS	Address and Data Input
	Data Output During Verify
P20–P22	Address Input
V_{DD}	Programming Power Supply
PROG	Program Pulse Input

WARNING

An attempt to program a missocketed 8749H (8748H) will result in severe damage to the part. An indication of a properly socketed part is the appearance of the ALE clock output. The lack of this clock may be used to disable the programmer.

The Program/Verify sequence is:

1) V_{DD} = 5V, Clock applied or internal oscillator operating. RESET = 0V, TEST 0 = 5V, EA = 5V, BUS and PROG floating. P10 and P11 must be tied to ground.
2) Insert 8749H (8748H) in programming socket.
3) TEST 0 = 0V (select program mode)
4) EA = 18V (activate program mode)
5) Address applied to BUS and P20–22
6) RESET = 5V (latch address)
7) Data applied to BUS
8) V_{DD} = 21V (programming power)
9) PROG = V_{CC} or float followed by one 50 ms pulse to 18V
10) V_{DD} = 5V
11) TEST 0 = 5V (verify mode)
12) Read and verify data on BUS
13) TEST 0 = 0V
14) RESET = 0V and repeat from step 5
15) Programmer should be at conditions of step 1 when 8749H (8748H) is removed from socket.

A.C. TIMING SPECIFICATION FOR PROGRAMMING 8748H/8749H ONLY

T_A = 25°C ±5°C; V_{CC} = 5V ±5%; V_{DD} = 21V ±0.5V

Symbol	Parameter	Min	Max	Unit	Test Conditions
t_{AW}	Address Setup Time to $\overline{RESET}$ ↑	$4t_{CY}$			
t_{WA}	Address Hold Time after $\overline{RESET}$ ↑	$4t_{CY}$			
t_{DW}	Data in Setup Time to PROG ↑	$4t_{CY}$			
t_{WD}	Data in Hold Time after PROG ↓	$4t_{CY}$			
t_{PH}	$\overline{RESET}$ Hold Time to Verify	$4t_{CY}$			
t_{VDDW}	V_{DD} Hold Time before PROG ↑	0	1.0	ms	
t_{VDDH}	V_{DD} Hold Time after PROG ↓	0	1.0	ms	
t_{PW}	Program Pulse Width	50	60	ms	
t_{TW}	TEST 0 Setup Time for Program Mode	$4t_{CY}$			
t_{WT}	TEST 0 Hold Time after Program Mode	$4t_{CY}$			
t_{DO}	TEST 0 to Data Out Delay		$4t_{CY}$		
t_{WW}	$\overline{RESET}$ Pulse Width to Latch Address	$4t_{CY}$			
t_r, t_f	V_{DD} and PROG Rise and Fall Times	0.5	100	μs	
t_{CY}	CPU Operation Cycle Time	3.75	5	μs	
t_{RE}	$\overline{RESET}$ Setup Time before EA ↑	$4t_{CY}$			

NOTE:
If TEST 0 is high, t_{DO} can be triggered by $\overline{RESET}$ ↑.

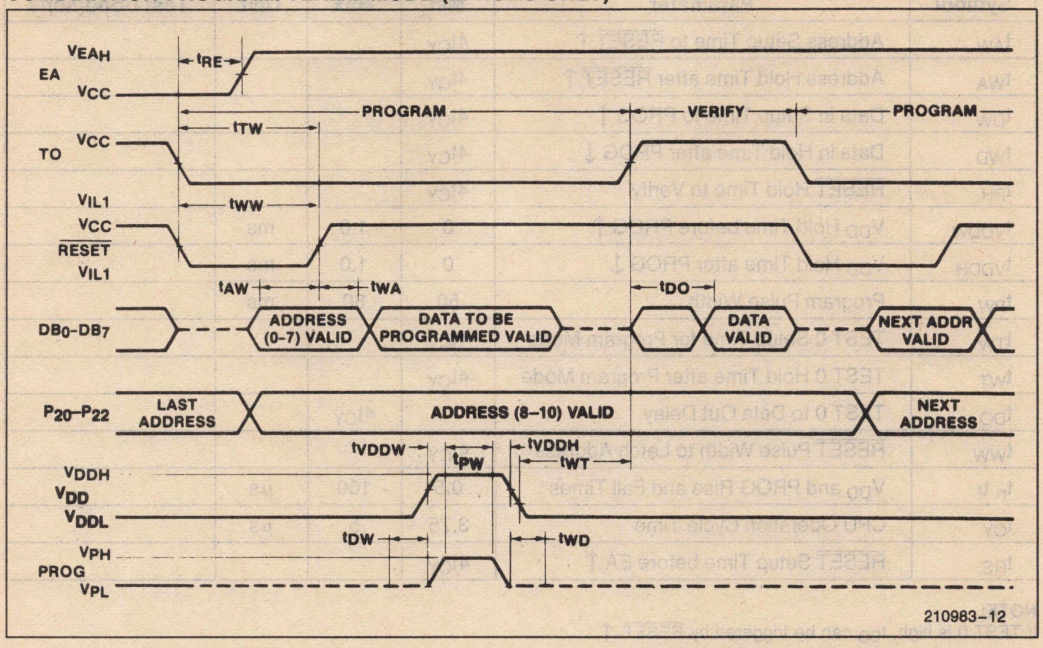
D.C. TIMING SPECIFICATION FOR PROGRAMMING 8748H/8749H ONLY

T_A = 25°C ±5°C; V_{CC} = 5V ±5%; V_{DD} = 21V ±0.5V

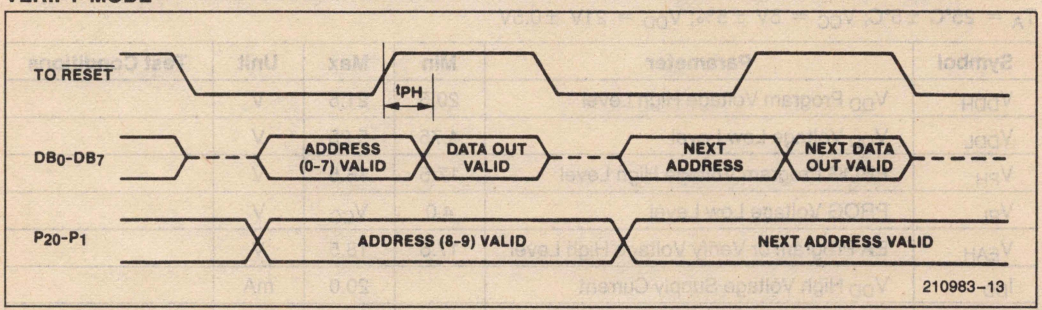
Symbol	Parameter	Min	Max	Unit	Test Conditions
V_{DDH}	V_{DD} Program Voltage High Level	20.5	21.5	V	
V_{DDL}	V_{DD} Voltage Low Level	4.75	5.25	V	
V_{PH}	PROG Program Voltage High Level	17.5	18.5	V	
V_{PL}	PROG Voltage Low Level	4.0	V_{CC}	V	
V_{EAH}	EA Program or Verify Voltage High Level	17.5	18.5	V	
I_{DD}	V_{DD} High Voltage Supply Current		20.0	mA	
I_{PROG}	PROG High Voltage Supply Current		1.0	mA	
I_{EA}	EA High Voltage Supply Current		1.0	mA	

WAVEFORMS

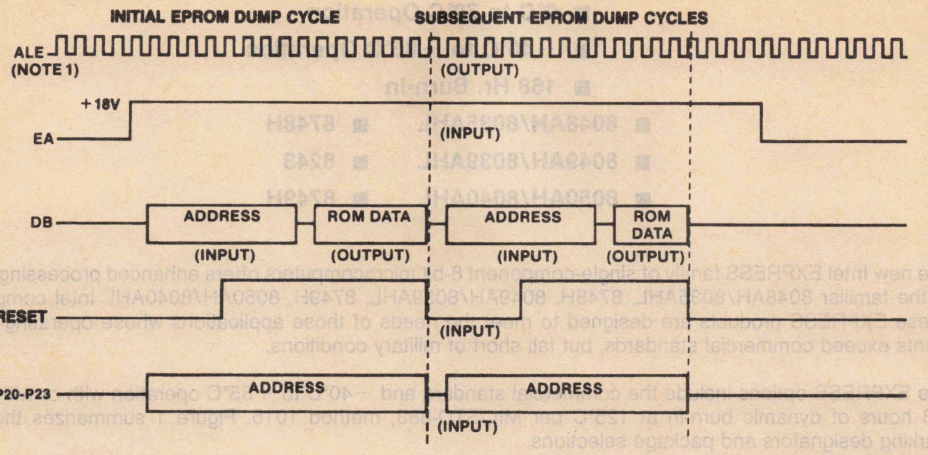
COMBINATION PROGRAM/VERIFY MODE (EPROMs ONLY)



VERIFY MODE



SUGGESTED EPROM VERIFICATION ALGORITHM FOR HMOS-E DEVICE ONLY



210983-14

	48H	49H
A10	0	ADDR
A11	0	0

$V_{CC} = V_{DD} = +5V$
$V_{SS} = 0V$

NOTE:
ALE is function of X1, X2 inputs.

MCS®-48
EXPRESS

- **0°C to 70°C Operation**
- **−40°C to +85°C Operation**
- **168 Hr. Burn-In**
- **8048AH/8035AHL**
- **8049AH/8039AHL**
- **8050AH/8040AHL**
- **8748H**
- **8243**
- **8749H**

The new Intel EXPRESS family of single-component 8-bit microcomputers offers enhanced processing options to the familiar 8048AH/8035AHL, 8748H, 8049AH/8039AHL, 8749H, 8050AH/8040AHL Intel components. These EXPRESS products are designed to meet the needs of those applications whose operating requirements exceed commercial standards, but fall short of military conditions.

The EXPRESS options include the commercial standard and −40°C to +85°C operation with or without 168 ±8 hours of dynamic burn-in at 125°C per MIL-STD-883, method 1015. Figure 1 summarizes the option marking designators and package selections.

For a complete description of 8048AH/8035AHL, 8748H, 8049AH/8309AHL, 8749H, 8040AHL and 8050AH features and operating characteristics, refer to the respective standard commercial grade data sheet. This document highlights only the electrical specifications which differ from the respective commercial part.

Temp Range °C	**0–70**	**−40–85**	**0–70**	**−40–85**
Burn In	**0 Hrs**	**0 Hrs**	**0 Hrs**	**0 Hrs**
	P8048AH	TP8048AH	QP8048AH	LP8048AH
	D8048AH	TD8048AH	QD8048AH	LD8048AH
	D8748H	TD8748H	QD8748H	LD8748H
	P8035AHL	TP8035AHL	QP8035AHL	LP8035AHL
	D8035AHL	TD8035AHL	QD8035AHL	LD8035AHL
	P8049AH	TP8049AH	QP8049AH	LP8049AH
	D8049AH	TD8049AH	QD8049AH	LD8049AH
	D8749A	TD8749AH	QD8749H	LD8749AH
	P8039AHL	TP8039AHL	QP8039AHL	LP8039AHL
	D8039AHL	TD8039AHL	QD8039AHL	LD8039AHL
	P8050AH	TP8050AH	QP8050AH	LP8050AH
	D8050AH	TD8050AH	QD8050AH	LD8050AH
	P8040AHL	TP8040AHL	QP8040AHL	LP8040AHL
	D8040AHL	TD8040AHL	QD8040AHL	LD8040AHL
	P8243	TP8243	QP8243	—
	D8243	TD8243	QD8243	LD8243

* Commercial Grade
P Plastic Package
D Cerdip Package

October 1986
Order Number: 270225-001

Extended Temperature Electrical Specification Deviations*

TP8048AH/TP8035AHL/LP8048AH/LP8035AHL
TD8048AH/TD8035AHL/LD8048AH/LD8035AHL

D.C. CHARACTERISTICS $T_A = -40°C$ to $+85°C$; $V_{CC} = V_{DD} = 5V \pm 10\%$; $V_{SS} = 0V$

Symbol	Parameter	Limits			Unit	Test Conditions
		Min	Typ	Max		
V_{IH}	Input High Voltage (All Except XTAL1, XTAL2, $\overline{RESET}$)	2.2		V_{CC}	V	
I_{DD}	V_{DD} Supply Current		4	8	mA	
$I_{DD} + I_{CC}$	Total Supply Current		40	80	mA	

TP8049AH/TP8039AHL/LP8049AH/LP8039AHL
TD8049AH/TD8039AHL/LD8049AH/LD8039AHL

D.C. CHARACTERISTICS $T_A = -40°C$ to $+85°C$; $V_{CC} = V_{DD} = 5V \pm 10\%$; $V_{SS} = 0V$

Symbol	Parameter	Limits			Unit	Test Conditions
		Min	Typ	Max		
V_{IH}	Input High Voltage (All Except XTAL1, XTAL2, $\overline{RESET}$)	2.2		V_{CC}	V	
I_{DD}	V_{DD} Supply Current		5	10	mA	
$I_{DD} + I_{CC}$	Total Supply Current		50	100	mA	

TP8050AH/TP8040AHL/LP8050AHL/LP8040AHL
TD8050AH/TD8040AHL/LD8050AH/LD8040AHL

D.C. CHARACTERISTICS $T_A = -40°C$ to $+85°C$; $V_{CC} = V_{DD} = 5V \pm 10\%$; $V_{SS} = 0V$

Symbol	Parameter	Limits			Unit	Test Conditions
		Min	Typ	Max		
V_{IH}	Input High Voltage (All Except XTAL1, XTAL2, $\overline{RESET}$)	2.2		V_{CC}	V	
I_{DD}	V_{DD} Supply Current		10	20	mA	
$I_{DD} + I_{CC}$	Total Supply Current		75	120	mA	

*Extended Temperature Electrical Specification Deviations**

TD8748H/LD8748H

D.C. CHARACTERISTICS $T_A = -40°C$ to $+85°C$; $V_{CC} = V_{DD} = 5V \pm 10\%$; $V_{SS} = 0V$

Symbol	Parameter	Limits			Unit	Test Conditions
		Min	Typ	Max		
V_{IH}	Input High Voltage (All Except XTAL1, XTAL2, $\overline{RESET}$)	2.2		V_{CC}	V	
$I_{DD} + I_{CC}$	Total Supply Current		50	130	mA	

TD8749H/LD8749H

D.C. CHARACTERISTICS $T_A = -40°C$ to $+85°C$; $V_{CC} = V_{DD} = 5V \pm 10\%$; $V_{SS} = 0V$

Symbol	Parameter	Limits			Unit	Test Conditions
		Min	Typ	Max		
V_{IH}	Input High Voltage (All Except XTAL1, XTAL2, $\overline{RESET}$)	2.2		V_{CC}	V	
$I_{DD} + I_{CC}$	Total Supply Current		75	150	mA	

TP8743/TD8243/LD8243

D.C. CHARACTERISTICS $T_A = -40°C$ to $+85°C$; $V_{CC} = 5V \pm 10\%$; $V_{SS} = 0V$

Symbol	Parameter	Limits			Unit	Test Conditions
		Min	Typ	Max		
I_{CC}	V_{CC} Supply Current		15	25	mA	

*Refer to individual commercial grade data sheet for complete operating characteristics.

MCS®-51 Architectural Overview

5

MCS®-51 Architectural Overview

ARCHITECTURAL OVERVIEW OF THE MCS®-51 FAMILY OF MICROCONTROLLERS

MEMBERS OF THE FAMILY

The MCS®-51 family of microcontrollers consists of the devices listed in Table 1. The basic architectural structure of these devices is shown in Figure 1.

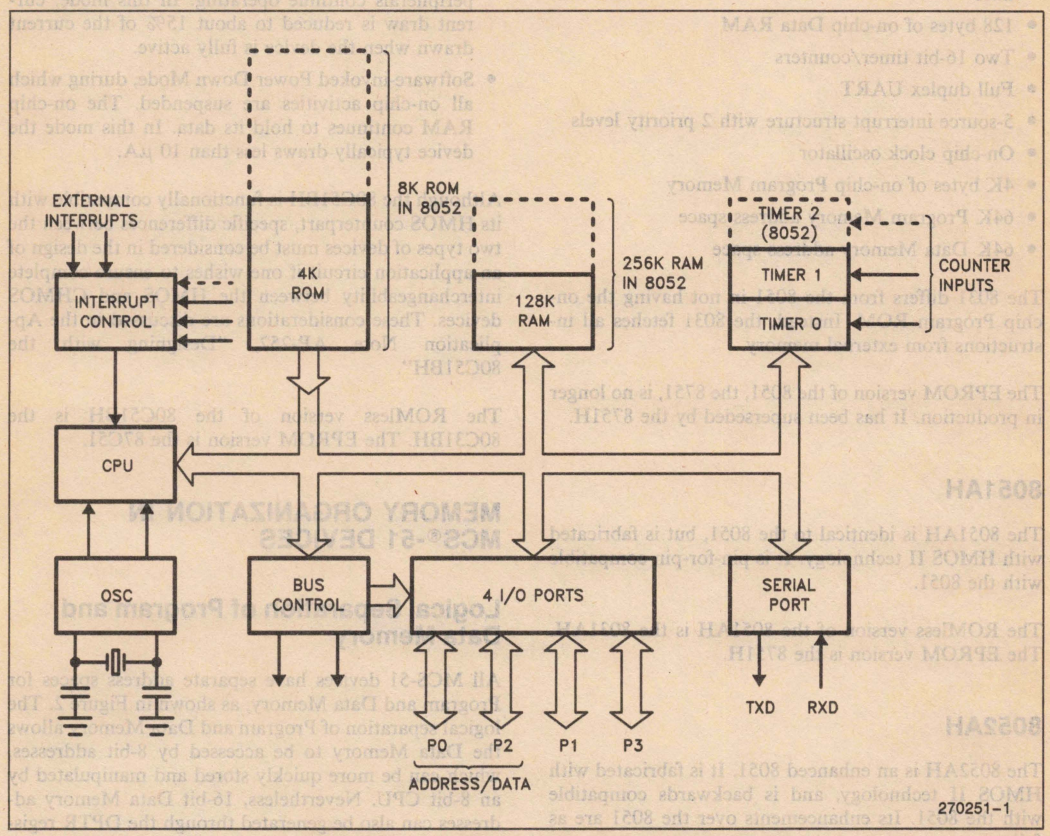


Figure 1. Block Diagram of the 8051/8052AH

Table 1. The MCS®-51 Family of Microcontrollers

Device Name	ROMless Version	EPROM Version	ROM Bytes	RAM Bytes	16-Bit Timers	Ckt Type
8051	8031	(8751)	4K	128	2	HMOS
8051AH	8031AH	8751H	4K	128	2	HMOS
8052AH	8032AH	8752BH	8K	256	3	HMOS
80C51BH	80C31BH	87C51	4K	128	2	CHMOS

8051

The 8051 is the original member of the MCS-51 Family, and has been in production since 1981. Among the features of the 8051 are:

- 8-bit CPU optimized for control applications
- Extensive Boolean processing (single-bit logic) capabilities
- 32 bidirectional and individually addressable I/O lines
- 128 bytes of on-chip Data RAM
- Two 16-bit timer/counters
- Full duplex UART
- 5-source interrupt structure with 2 priority levels
- On-chip clock oscillator
- 4K bytes of on-chip Program Memory
- 64K Program Memory address space
- 64K Data Memory address space

The 8031 differs from the 8051 in not having the on-chip Program ROM. Instead, the 8031 fetches all instructions from external memory.

The EPROM version of the 8051, the 8751, is no longer in production. It has been superseded by the 8751H.

8051AH

The 8051AH is identical to the 8051, but is fabricated with HMOS II technology. It is pin-for-pin compatible with the 8051.

The ROMless version of the 8051AH is the 8031AH. The EPROM version is the 8751H.

8052AH

The 8052AH is an enhanced 8051. It is fabricated with HMOS II technology, and is backwards compatible with the 8051. Its enhancements over the 8051 are as follows:

- 256 bytes of on-chip Data RAM
- Three timer/counters
- 6-source interrupt structure
- 8K bytes of on-chip Program ROM

The ROMless version of the 8052AH is the 8032AH. The EPROM version is the 8752BH.

A separate product, the 8052AH-BASIC, is an 8052AH with a full BASIC interpreter in the on-chip ROM.

80C51BH

The 80C51BH is the CHMOS version of the 8051. Functionally, it is fully compatible with the 8051, but being CMOS it draws less current than its HMOS counterpart. To further exploit the power savings available in CMOS circuitry, two reduced power modes are added:

- Software-invoked Idle Mode, during which the CPU is turned off while the RAM and other on-chip peripherals continue operating. In this mode, current draw is reduced to about 15% of the current drawn when the device is fully active.
- Software-invoked Power Down Mode, during which all on-chip activities are suspended. The on-chip RAM continues to hold its data. In this mode the device typically draws less than 10 μA.

Although the 80C51BH is functionally compatible with its HMOS counterpart, specific differences between the two types of devices must be considered in the design of an application circuit if one wishes to ensure complete interchangeability between the HMOS and CHMOS devices. These considerations are discussed in the Application Note AP-252, "Designing with the 80C51BH".

The ROMless version of the 80C51BH is the 80C31BH. The EPROM version is the 87C51.

MEMORY ORGANIZATION IN MCS®-51 DEVICES

Logical Separation of Program and Data Memory

All MCS-51 devices have separate address spaces for Program and Data Memory, as shown in Figure 2. The logical separation of Program and Data Memory allows the Data Memory to be accessed by 8-bit addresses, which can be more quickly stored and manipulated by an 8-bit CPU. Nevertheless, 16-bit Data Memory addresses can also be generated through the DPTR register.

Program Memory can only be read, not written to. There can be up to 64K bytes of Program Memory. In the 8051, 8051AH, 80C51BH, and their EPROM versions, the lowest 4K bytes of Program Memory are on-chip. The 8052AH provides 8K bytes of on-chip Program Memory storage. In the ROMless versions (8031, 8031AH, 80C31BH, 8032AH) all Program Memory is external. The read strobe for external Program Memory is the signal $\overline{\text{PSEN}}$ (Program Store Enable).

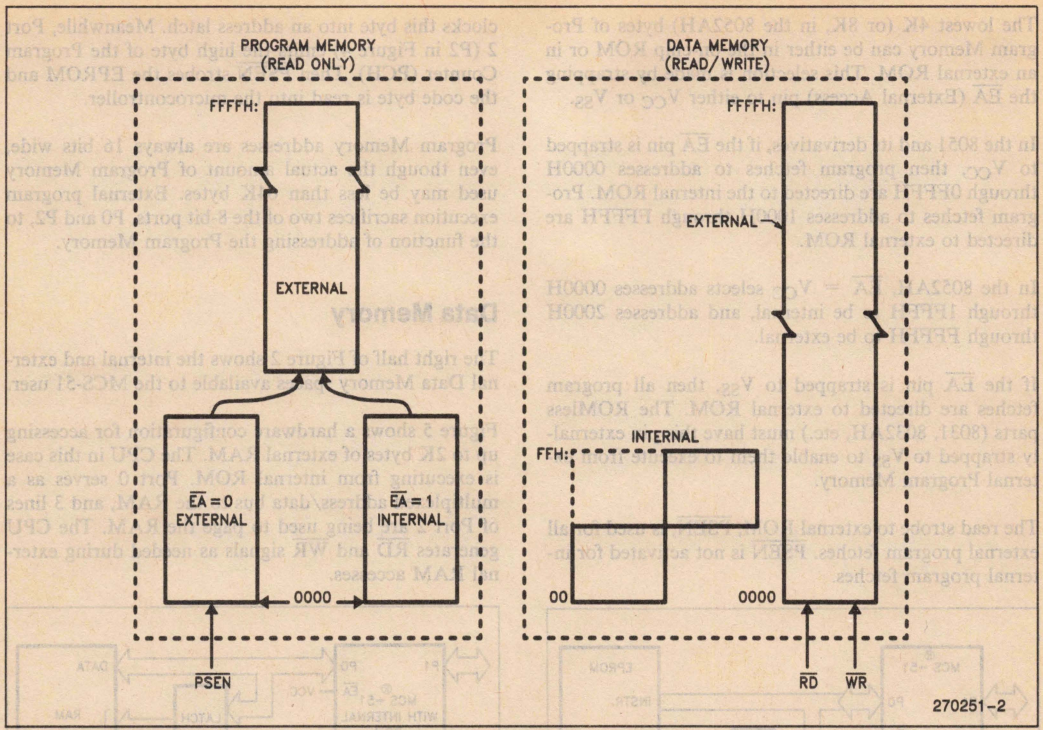


Figure 2. MCS®-51 Memory Structure

Data Memory occupies a separate address space from Program Memory. Up to 64K bytes of external RAM can be addressed in the external Data Memory space. The CPU generates read and write signals, $\overline{RD}$ and $\overline{WR}$, as needed during external Data Memory accesses.

External Program Memory and external Data Memory may be combined if desired by applying the $\overline{RD}$ and $\overline{PSEN}$ signals to the inputs of an AND gate and using the output of the gate as the read strobe to the external Program/Data memory.

Program Memory

Figure 3 shows a map of the lower part of the Program Memory. After reset, the CPU begins execution from location 0000H.

As shown in Figure 3, each interrupt is assigned a fixed location in Program Memory. The interrupt causes the CPU to jump to that location, where it commences execution of the service routine. External Interrupt 0, for example, is assigned to location 0003H. If External Interrupt 0 is going to be used, its service routine must begin at location 0003H. If the interrupt is not going to be used, its service location is available as general purpose Program Memory.

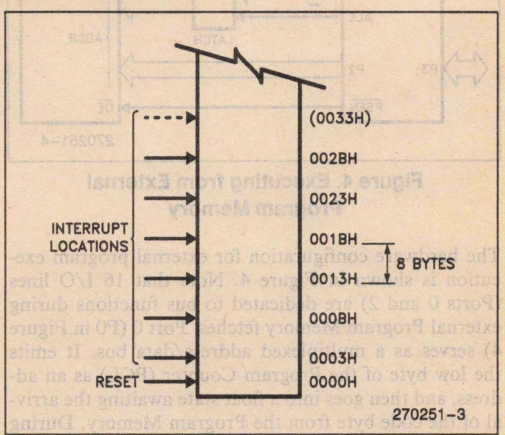


Figure 3. MCS®-51 Program Memory

The interrupt service locations are spaced at 8-byte intervals: 0003H for External Interrupt 0, 000BH for Timer 0, 0013H for External Interrupt 1, 001BH for Timer 1, etc. If an interrupt service routine is short enough (as is often the case in control applications), it can reside entirely within that 8-byte interval. Longer service routines can use a jump instruction to skip over subsequent interrupt locations, if other interrupts are in use.

The lowest 4K (or 8K, in the 8052AH) bytes of Program Memory can be either in the on-chip ROM or in an external ROM. This selection is made by strapping the $\overline{EA}$ (External Access) pin to either V_{CC} or V_{SS}.

In the 8051 and its derivatives, if the $\overline{EA}$ pin is strapped to V_{CC}, then program fetches to addresses 0000H through 0FFFH are directed to the internal ROM. Program fetches to addresses 1000H through FFFFH are directed to external ROM.

In the 8052AH, $\overline{EA} = V_{CC}$ selects addresses 0000H through 1FFFH to be internal, and addresses 2000H through FFFFH to be external.

If the $\overline{EA}$ pin is strapped to V_{SS}, then all program fetches are directed to external ROM. The ROMless parts (8031, 8032AH, etc.) must have this pin externally strapped to V_{SS} to enable them to execute from external Program Memory.

The read strobe to external ROM, $\overline{PSEN}$, is used for all external program fetches. $\overline{PSEN}$ is not activated for internal program fetches.

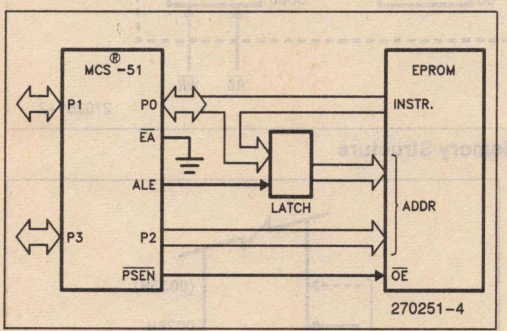


Figure 4. Executing from External Program Memory

The hardware configuration for external program execution is shown in Figure 4. Note that 16 I/O lines (Ports 0 and 2) are dedicated to bus functions during external Program Memory fetches. Port 0 (P0 in Figure 4) serves as a multiplexed address/data bus. It emits the low byte of the Program Counter (PCL) as an address, and then goes into a float state awaiting the arrival of the code byte from the Program Memory. During the time that the low byte of the Program Counter is valid on P0, the signal ALE (Address Latch Enable) clocks this byte into an address latch. Meanwhile, Port 2 (P2 in Figure 4) emits the high byte of the Program Counter (PCH). Then $\overline{PSEN}$ strobes the EPROM and the code byte is read into the microcontroller.

Program Memory addresses are always 16 bits wide, even though the actual amount of Program Memory used may be less than 64K bytes. External program execution sacrifices two of the 8-bit ports, P0 and P2, to the function of addressing the Program Memory.

Data Memory

The right half of Figure 2 shows the internal and external Data Memory spaces available to the MCS-51 user.

Figure 5 shows a hardware configuration for accessing up to 2K bytes of external RAM. The CPU in this case is executing from internal ROM. Port 0 serves as a multiplexed address/data bus to the RAM, and 3 lines of Port 2 are being used to page the RAM. The CPU generates $\overline{RD}$ and $\overline{WR}$ signals as needed during external RAM accesses.

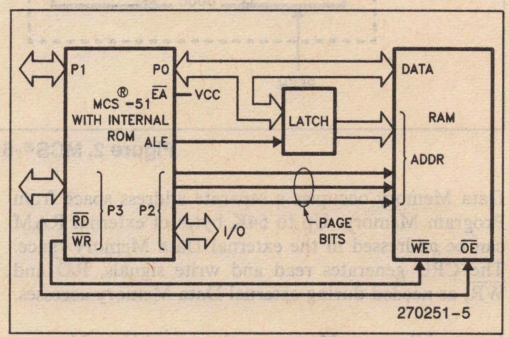


Figure 5. Accessing External Data Memory. If the Program Memory is Internal, the Other Bits of P2 are Available as I/O.

There can be up to 64K bytes of external Data Memory. External Data Memory addresses can be either 1 or 2 bytes wide. One-byte addresses are often used in conjunction with one or more other I/O lines to page the RAM, as shown in Figure 5. Two-byte addresses can also be used, in which case the high address byte is emitted at Port 2.

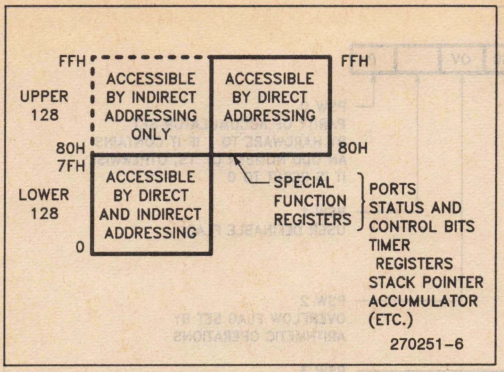


Figure 6. Internal Data Memory

Internal Data Memory is mapped in Figure 6. The memory space is shown divided into three blocks, which are generally referred to as the Lower 128, the Upper 128, and SFR space.

Internal Data Memory addresses are always one byte wide, which implies an address space of only 256 bytes. However, the addressing modes for internal RAM can in fact accommodate 384 bytes, using a simple trick. Direct addresses higher than 7FH access one memory space, and indirect addresses higher than 7FH access a different memory space. Thus Figure 6 shows the Upper 128 and SFR space occupying the same block of addresses, 80H through FFH, although they are physically separate entities.

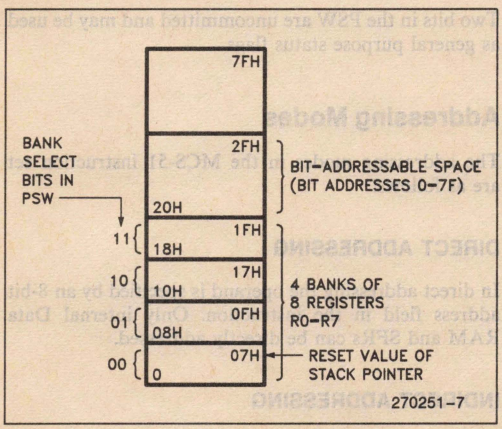


Figure 7. The Lower 128 Bytes of Internal RAM

The Lower 128 bytes of RAM are present in all MCS-51 devices as mapped in Figure 7. The lowest 32 bytes are grouped into 4 banks of 8 registers. Program instructions call out these registers as R0 through R7. Two bits in the Program Status Word (PSW) select which register bank is in use. This allows more efficient use of code space, since register instructions are shorter than instructions that use direct addressing.

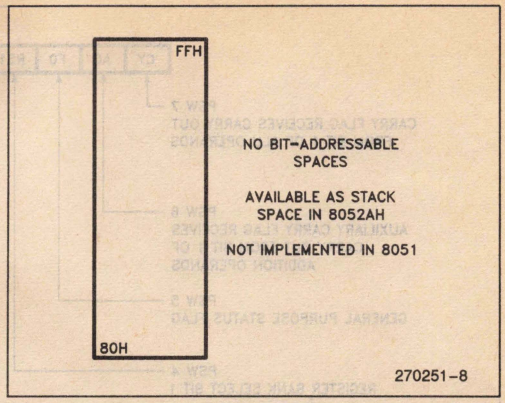


Figure 8. The Upper 128 Bytes of Internal RAM

The next 16 bytes above the register banks form a block of bit-addressable memory space. The MCS-51 instruction set includes a wide selection of single-bit instructions, and the 128 bits in this area can be directly addressed by these instructions. The bit addresses in this area are 00H through 7FH.

All of the bytes in the Lower 128 can be accessed by either direct or indirect addressing. The Upper 128 (Figure 8) can only be accessed by indirect addressing. The Upper 128 bytes of RAM are not implemented in the 8051, but are in the 8052AH.

Figure 9 gives a brief look at the Special Function Register (SFR) space. SFRs include the Port latches, timers, peripheral controls, etc. These registers can only be accessed by direct addressing. In general, all MCS-51 microcontrollers have the same SFRs as the 8051, and at the same addresses in SFR space. However, enhancements to the 8051 have additional SFRs that are not present in the 8051, nor perhaps in other proliferations of the family.

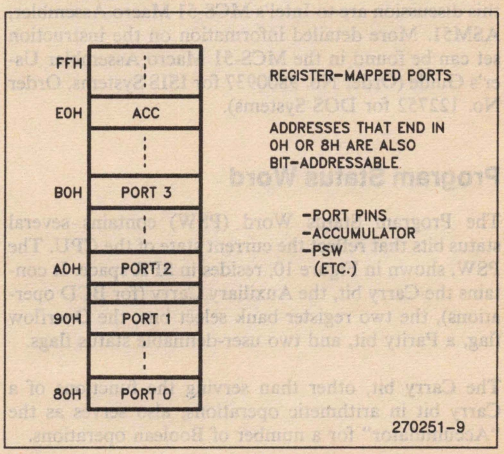


Figure 9. SFR Space

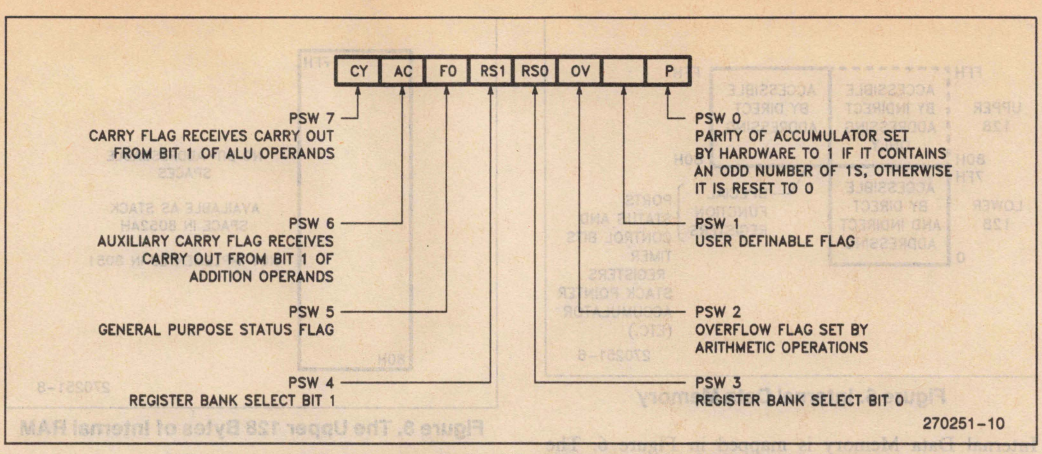


Figure 10. PSW (Program Status Word) Register in MCS®-51 Devices

Sixteen addresses in SFR space are both byte- and bit-addressable. The bit-addressable SFRs are those whose address ends in 000B. The bit addresses in this area are 80H through FFH.

THE MCS®-51 INSTRUCTION SET

All members of the MCS-51 family execute the same instruction set. The MCS-51 instruction set is optimized for 8-bit control applications. It provides a variety of fast addressing modes for accessing the internal RAM to facilitate byte operations on small data structures. The instruction set provides extensive support for one-bit variables as a separate data type, allowing direct bit manipulation in control and logic systems that require Boolean processing.

An overview of the MCS-51 instruction set is presented below, with a brief description of how certain instructions might be used. References to "the assembler" in this discussion are to Intel's MCS-51 Macro Assembler, ASM51. More detailed information on the instruction set can be found in the MCS-51 Macro Assembler User's Guide (Order No. 9800937 for ISIS Systems, Order No. 122752 for DOS Systems).

Program Status Word

The Program Status Word (PSW) contains several status bits that reflect the current state of the CPU. The PSW, shown in Figure 10, resides in SFR space. It contains the Carry bit, the Auxiliary Carry (for BCD operations), the two register bank select bits, the Overflow flag, a Parity bit, and two user-definable status flags.

The Carry bit, other than serving the functions of a Carry bit in arithmetic operations, also serves as the "Accumulator" for a number of Boolean operations.

The bits RS0 and RS1 are used to select one of the four register banks shown in Figure 7. A number of instructions refer to these RAM locations as R0 through R7. The selection of which of the four banks is being referred to is made on the basis of the bits RS0 and RS1 at execution time.

The Parity bit reflects the number of 1s in the Accumulator: P = 1 if the Accumulator contains an odd number of 1s, and P = 0 if the Accumulator contains an even number of 1s. Thus the number of 1s in the Accumulator plus P is always even.

Two bits in the PSW are uncommitted and may be used as general purpose status flags.

Addressing Modes

The addressing modes in the MCS-51 instruction set are as follows:

DIRECT ADDRESSING

In direct addressing the operand is specified by an 8-bit address field in the instruction. Only internal Data RAM and SFRs can be directly addressed.

INDIRECT ADDRESSING

In indirect addressing the instruction specifies a register which contains the address of the operand. Both internal and external RAM can be indirectly addressed.

The address register for 8-bit addresses can be R0 or R1 of the selected register bank, or the Stack Pointer. The address register for 16-bit addresses can only be the 16-bit "data pointer" register, DPTR.

REGISTER INSTRUCTIONS

The register banks, containing registers R0 through R7, can be accessed by certain instructions which carry a 3-bit register specification within the opcode of the instruction. Instructions that access the registers this way are code efficient, since this mode eliminates an address byte. When the instruction is executed, one of the eight registers in the selected bank is accessed. One of four banks is selected at execution time by the two bank select bits in the PSW.

REGISTER-SPECIFIC INSTRUCTIONS

Some instructions are specific to a certain register. For example, some instructions always operate on the Accumulator, or Data Pointer, etc., so no address byte is needed to point to it. The opcode itself does that. Instructions that refer to the Accumlator as A assemble as accumulator-specific opcodes.

IMMEDIATE CONSTANTS

The value of a constant can follow the opcode in Program Memory. For example,

```
MOV A, #100
```

loads the Accumulator with the decimal number 100. The same number could be specified in hex digits as 64H.

INDEXED ADDRESSING

Only Program Memory can be accessed with indexed addressing, and it can only be read. This addressing mode is intended for reading look-up tables in Program Memory. A 16-bit base register (either DPTR or the Program Counter) points to the base of the table, and the Accumulator is set up with the table entry number. The address of the table entry in Program Memory is formed by adding the Accumulator data to the base pointer.

Another type of indexed addressing is used in the "case jump" instruction. In this case the destination address of a jump instruction is computed as the sum of the base pointer and the Accumulator data.

Arithmetic Instructions

The menu of arithmetic instructions is listed in Table 2. The table indicates the addressing modes that can be used with each instruction to access the <byte> operand. For example, the ADD A,<byte> instruction can be written as:

```
ADD    A,7FH     (direct addressing)
ADD    A,@R0     (indirect addressing)
ADD    A,R7      (register addressing)
ADD    A,#127    (immediate constant)
```

The execution times listed in Table 2 assume a 12 MHz clock frequency. All of the arithmetic instructions execute in 1 μs except the INC DPTR instruction, which takes 2 μs, and the Multiply and Divide instructions, which take 4 μs.

Table 2. A List of the MCS®-51 Arithmetic Instructions

Mnemonic	Operation	Addressing Modes				Execution Time (μs)
		Dir	Ind	Reg	Imm	
ADD A,<byte>	A = A + <byte>	X	X	X	X	1
ADDC A,<byte>	A = A + <byte> + C	X	X	X	X	1
SUBB A,<byte>	A = A − <byte> − C	X	X	X	X	1
INC A	A = A + 1	Accumulator only				1
INC <byte>	<byte> = <byte> + 1	X	X	X		1
INC DPTR	DPTR = DPTR + 1	Data Pointer only				2
DEC A	A = A − 1	Accumulator only				1
DEC <byte>	<byte> = <byte> − 1	X	X	X		1
MUL AB	B:A = B x A	ACC and B only				4
DIV AB	A = Int [A/B] B = Mod [A/B]	ACC and B only				4
DA A	Decimal Adjust	Accumulator only				1

Note that any byte in the internal Data Memory space can be incremented or decremented without going through the Accumulator.

One of the INC instructions operates on the 16-bit Data Pointer. The Data Pointer is used to generate 16-bit addresses for external memory, so being able to increment it in one 16-bit operation is a useful feature.

The MUL AB instruction multiplies the Accumulator by the data in the B register and puts the 16-bit product into the concatenated B and Accumulator registers.

The DIV AB instruction divides the Accumulator by the data in the B register and leaves the 8-bit quotient in the Accumulator, and the 8-bit remainder in the B register.

Oddly enough, DIV AB finds less use in arithmetic "divide" routines than in radix conversions and programmable shift operations. An example of the use of DIV AB in a radix conversion will be given later. In shift operations, dividing a number by 2^n shifts its n bits to the right. Using DIV AB to perform the division completes the shift in 4 μs and leaves the B register holding the bits that were shifted out.

The DA A instruction is for BCD arithmetic operations. In BCD arithmetic, ADD and ADDC instructions should always be followed by a DA A operation, to ensure that the result is also in BCD. Note that DA A will not convert a binary number to BCD. The DA A operation produces a meaningful result only as the second step in the addition of two BCD bytes.

Logical Instructions

Table 3 shows the list of MCS-51 logical instructions. The instructions that perform Boolean operations (AND, OR, Exclusive OR, NOT) on bytes perform the operation on a bit-by-bit basis. That is, if the Accumulator contains 00110101B and <byte> contains 01010011B, then

```
ANL     A,<byte>
```

will leave the Accumulator holding 00010001B.

The addressing modes that can be used to access the <byte> operand are listed in Table 3. Thus, the ANL A,<byte> instruction may take any of the forms

```
ANL     A,7FH       (direct addressing)
ANL     A,@R1       (indirect addressing)
ANL     A,R6        (register addressing)
ANL     A,#53H      (immediate constant)
```

All of the logical instructions that are Accumulator-specific execute in 1μs (using a 12 MHz clock). The others take 2 μs.

Table 3. A List of the MCS®-51 Logical Instructions

Mnemonic	Operation	Addressing Modes				Execution Time (μs)
		Dir	Ind	Reg	Imm	
ANL A,<byte>	A = A .AND. <byte>	X	X	X	X	1
ANL <byte>,A	<byte> = <byte> .AND. A	X				1
ANL <byte>,#data	<byte> = <byte> .AND. #data	X				2
ORL A,<byte>	A = A .OR. <byte>	X	X	X	X	1
ORL <byte>,A	<byte> = <byte> .OR. A	X				1
ORL <byte>,#data	<byte> = <byte> .OR. #data	X				2
XRL A,<byte>	A = A .XOR. <byte>	X	X	X	X	1
XRL <byte>,A	<byte> = <byte> .XOR. A	X				1
XRL <byte>,#data	<byte> = <byte> .XOR. #data	X				2
CRL A	A = 00H	Accumulator only				1
CPL A	A = .NOT. A	Accumulator only				1
RL A	Rotate ACC Left 1 bit	Accumulator only				1
RLC A	Rotate Left through Carry	Accumulator only				1
RR A	Rotate ACC Right 1 bit	Accumulator only				1
RRC A	Rotate Right through Carry	Accumulator only				1
SWAP A	Swap Nibbles in A	Accumulator only				1

Note that Boolean operations can be performed on any byte in the internal Data Memory space without going through the Accumulator. The XRL <byte>, #data instruction, for example, offers a quick and easy way to invert port bits, as in

```
XRL    P1,#0FFH
```

If the operation is in response to an interrupt, not using the Accumulator saves the time and effort to stack it in the service routine.

The Rotate instructions (RL A, RLC A, etc.) shift the Accumulator 1 bit to the left or right. For a left rotation, the MSB rolls into the LSB position. For a right rotation, the LSB rolls into the MSB position.

The SWAP A instruction interchanges the high and low nibbles within the Accumulator. This is a useful operation in BCD manipulations. For example, if the Accumulator contains a binary number which is known to be less than 100, it can be quickly converted to BCD by the following code:

```
MOV   B,#10
DIV   AB
SWAP  A
ADD   A,B
```

Dividing the number by 10 leaves the tens digit in the low nibble of the Accumulator, and the ones digit in the B register. The SWAP and ADD instructions move the tens digit to the high nibble of the Accumulator, and the ones digit to the low nibble.

Data Transfers

INTERNAL RAM

Table 4 shows the menu of instructions that are available for moving data around within the internal memory spaces, and the addressing modes that can be used with each one. With a 12 MHz clock, all of these instructions execute in either 1 or 2 μs.

The MOV <dest>, <src> instruction allows data to be transferred between any two internal RAM or SFR locations without going through the Accumulator. Remember the Upper 128 byes of data RAM can be accessed only by indirect addressing, and SFR space only by direct addressing.

Note that in all MCS-51 devices, the stack resides in on-chip RAM, and grows upwards. The PUSH instruction first increments the Stack Pointer (SP), then copies the byte into the stack. PUSH and POP use only direct addressing to identify the byte being saved or restored, but the stack itself is accessed by indirect addressing using the SP register. This means the stack can go into the Upper 128, if they are implemented, but not into SFR space.

The Upper 128 are not implemented in the 8051, 8051AH, or 80C51BH, nor in their ROMless or EPROM counterparts. With these devices, if the SP points to the Upper 128, PUSHed bytes are lost, and POPped bytes are indeterminate.

The Data Transfer instructions include a 16-bit MOV that can be used to initialize the Data Pointer (DPTR) for look-up tables in Program Memory, or for 16-bit external Data Memory accesses.

The XCH A, <byte> instruction causes the Accumulator and addressed byte to exchange data. The XCHD A,@Ri instruction is similar, but only the low nibbles are involved in the exchange.

To see how XCH and XCHD can be used to facilitate data manipulations, consider first the problem of shifting an 8-digit BCD number two digits to the right. Figure 11 shows how this can be done using direct

Table 4. A List of the MCS®-51 Data Transfer Instructions that Access Internal Data Memory Space

Mnemonic	Operation	Addressing Modes				Execution Time (μs)
		Dir	Ind	Reg	Imm	
MOV A,<src>	A = <src>	X	X	X	X	1
MOV <dest>,A	<dest> = A	X	X	X		1
MOV <dest>, <src>	<dest> = <src>	X	X	X	X	2
MOV DPTR,#data16	DPTR = 16-bit immediate constant.				X	2
PUSH <src>	INC SP : MOV "@SP",<src>	X				2
POP <dest>	MOV <dest>, "@SP" : DEC SP	X				2
XCH A,<byte>	ACC and <byte> exchange data	X	X	X		1
XCHD A,@Ri	ACC and @Ri exchange low nibbles		X			1

		2A	2B	2C	2D	2E	ACC
MOV	A,2EH	00	12	34	56	78	78
MOV	2EH,2DH	00	12	34	56	56	78
MOV	2DH,2CH	00	12	34	34	56	78
MOV	2CH,2BH	00	12	12	34	56	78
MOV	2BH,#0	00	00	12	34	56	78

(a) Using direct MOVs: 14 bytes, 9 μs

		2A	2B	2C	2D	2E	ACC
CLR	A	00	12	34	56	78	00
XCH	A,2BH	00	00	34	56	78	12
XCH	A,2CH	00	00	12	56	78	34
XCH	A,2DH	00	00	12	34	78	56
XCH	A,2EH	00	00	12	34	56	78

(b) Using XCHs: 9 bytes, 5 μs

Figure 11. Shifting a BCD Number Two Digits to the Right

MOVs, and for comparison how it can be done using XCH instructions. To aid in understanding how the code works, the contents of the registers that are holding the BCD number and the content of the Accumulator are shown alongside each instruction to indicate their status after the instruction has been executed.

After the routine has been executed, the Accumulator contains the two digits that were shifted out on the right. Doing the routine with direct MOVs uses 14 code bytes and 9 μs of execution time (assuming a 12 MHz clock). The same operation with XCHs uses less code and executes almost twice as fast.

To right-shift by an odd number of digits, a one-digit shift must be executed. Figure 12 shows a sample of code that will right-shift a BCD number one digit, using the XCHD instruction. Again, the contents of the registers holding the number and of the Accumulator are shown alongside each instruction.

			2A	2B	2C	2D	2E	ACC
	MOV	R1,#2EH	00	12	34	56	78	XX
	MOV	R0,#2DH	00	12	34	56	78	XX
	loop for R1 = 2EH:							
LOOP:	MOV	A,@R1	00	12	34	56	78	78
	XCHD	A,@R0	00	12	34	58	78	76
	SWAP	A	00	12	34	58	78	67
	MOV	@R1,A	00	12	34	58	67	67
	DEC	R1	00	12	34	58	67	67
	DEC	R0	00	12	34	58	67	67
	CJNE	R1,#2AH,LOOP						
	loop for R1 = 2DH:		00	12	38	45	67	45
	loop for R1 = 2CH:		00	18	23	45	67	23
	loop for R1 = 2BH:		08	01	23	45	67	01
	CLR	A	08	01	23	45	67	00
	XCH	A,2AH	00	01	23	45	67	08

Figure 12. Shifting a BCD Number One Digit to the Right

First, pointers R1 and R0 are set up to point to the two bytes containing the last four BCD digits. Then a loop is executed which leaves the last byte, location 2EH, holding the last two digits of the shifted number. The pointers are decremented, and the loop is repeated for location 2DH. The CJNE instruction (Compare and Jump if Not Equal) is a loop control that will be described later.

The loop is executed from LOOP to CJNE for R1 = 2EH, 2DH, 2CH and 2BH. At that point the digit that was originally shifted out on the right has propagated to location 2AH. Since that location should be left with 0s, the lost digit is moved to the Accumulator.

EXTERNAL RAM

Table 5 shows a list of the Data Transfer instructions that access external Data Memory. Only indirect addressing can be used. The choice is whether to use a one-byte address, @Ri, where Ri can be either R0 or

R1 of the selected register bank, or a two-byte address, @DPTR. The disadvantage to using 16-bit addresses if only a few K bytes of external RAM are involved is that 16-bit addresses use all 8 bits of Port 2 as address bus. On the other hand, 8-bit addresses allow one to address a few K bytes of RAM, as shown in Figure 5, without having to sacrifice all of Port 2.

All of these instructions execute in 2 μs, with a 12 MHz clock.

Table 5. A List of the MCS®-51 Data Transfer Instructions that Access External Data Memory Space

Address Width	Mnemonic	Operation	Execution Time (μs)
8 bits	MOVX A,@Ri	Read external RAM @Ri	2
8 bits	MOVX @Ri,A	Write external RAM @Ri	2
16 bits	MOVX A,@DPTR	Read external RAM @DPTR	2
16 bits	MOVX @DPTR,A	Write external RAM @DPTR	2

Note that in all external Data RAM accesses, the Accumulator is always either the destination or source of the data.

The read and write strobes to external RAM are activated only during the execution of a MOVX instruction. Normally these signals are inactive, and in fact if they're not going to be used at all, their pins are available as extra I/O lines. More about that later.

LOOKUP TABLES

Table 6 shows the two instructions that are available for reading lookup tables in Program Memory. Since these instructions access only Program Memory, the lookup tables can only be read, not updated. The mnemonic is MOVC for "move constant".

If the table access is to external Program Memory, then the read strobe is $\overline{\text{PSEN}}$.

Table 6. The MCS®-51 Lookup Table Read Instructions

Mnemonic	Operation	Execution Time (μs)
MOVC A,@A+DPTR	Read Pgm Memory at (A+DPTR)	2
MOVC A,@A+PC	Read Pgm Memory at (A+PC)	2

The first MOVC instruction in Table 6 can accommodate a table of up to 256 entries, numbered 0 through 255. The number of the desired entry is loaded into the Accumulator, and the Data Pointer is set up to point to beginning of the table. Then

```
MOVC    A,@A+DPTR
```

copies the desired table entry into the Accumulator.

The other MOVC instruction works the same way, except the Program Counter (PC) is used as the table base, and the table is accessed through a subroutine. First the number of the desired entry is loaded into the Accumulator, and the subroutine is called:

```
        MOV     A,ENTRY_NUMBER
        CALL    TABLE
```

The subroutine "TABLE" would look like this:

```
TABLE:  MOVC    A,@A+PC
        RET
```

The table itself immediately follows the RET (return) instruction in Program Memory. This type of table can have up to 255 entries, numbered 1 through 255. Number 0 can not be used, because at the time the MOVC instruction is executed, the PC contains the address of the RET instruction. An entry numbered 0 would be the RET opcode itself.

Boolean Instructions

MCS-51 devices contain a complete Boolean (single-bit) processor. The internal RAM contains 128 addressable bits, and the SFR space can support up to 128 other addressable bits. All of the port lines are bit-addressable, and each one can be treated as a separate single-bit port. The instructions that access these bits are not just conditional branches, but a complete menu of move, set, clear, complement, OR, and AND instructions. These kinds of bit operations are not easily obtained in other architectures with any amount of byte-oriented software.

Table 7. A List of the MCS®-51 Boolean Instructions

Mnemonic		Operation	Execution Time (μs)
ANL	C,bit	C = C .AND. bit	2
ANL	C,/bit	C = C .AND. .NOT. bit	2
ORL	C,bit	C = C .OR. bit	2
ORL	C,/bit	C = C .OR. .NOT. bit	2
MOV	C,bit	C = bit	1
MOV	bit,C	bit = C	2
CLR	C	C = 0	1
CLR	bit	bit = 0	1
SETB	C	C = 1	1
SETB	bit	bit = 1	1
CPL	C	C = .NOT. C	1
CPL	bit	bit = .NOT. bit	1
JC	rel	Jump if C = 1	2
JNC	rel	Jump if C = 0	2
JB	bit,rel	Jump if bit = 1	2
JNB	bit,rel	Jump if bit = 0	2
JBC	bit,rel	Jump if bit = 1; CLR bit	2

The instruction set for the Boolean processor is shown in Table 7. All bit accesses are by direct addressing. Bit addresses 00H through 7FH are in the Lower 128, and bit addresses 80H through FFH are in SFR space.

Note how easily an internal flag can be moved to a port pin:

```
MOV     C,FLAG
MOV     P1.0,C
```

In this example, FLAG is the name of any addressable bit in the Lower 128 or SFR space. An I/O line (the LSB of Port 1, in this case) is set or cleared depending on whether the flag bit is 1 or 0.

The Carry bit in the PSW is used as the single-bit Accumulator of the Boolean processor. Bit instructions that refer to the Carry bit as C assemble as Carry-specific instructions (CLR C, etc). The Carry bit also has a direct address, since it resides in the PSW register, which is bit-addressable.

Note that the Boolean instruction set includes ANL and ORL operations, but not the XRL (Exclusive OR) operation. An XRL operation is simple to implement in software. Suppose, for example, it is required to form the Exclusive OR of two bits:

```
C = bit1 .XRL. bit2
```

The software to do that could be as follows:

```
        MOV     C,bit1
        JNB     bit2,OVER
        CPL     C
OVER:   (continue)
```

First, bit1 is moved to the Carry. If bit2 = 0, then C now contains the correct result. That is, bit1 .XRL. bit2 = bit1 if bit2 = 0. On the other hand, if bit2 = 1 C now contains the complement of the correct result. It need only be inverted (CPL C) to complete the operation.

This code uses the JNB instruction, one of a series of bit-test instructions which execute a jump if the addressed bit is set (JC, JB, JBC) or if the addressed bit is not set (JNC, JNB). In the above case, bit2 is being tested, and if bit2 = 0 the CPL C instruction is jumped over.

JBC executes the jump if the addressed bit is set, and also clears the bit. Thus a flag can be tested and cleared in one operation.

All the PSW bits are directly addressable, so the Parity bit, or the general purpose flags, for example, are also available to the bit-test instructions.

RELATIVE OFFSET

The destination address for these jumps is specified to the assembler by a label or by an actual address in Program Memory. However, the destination address assembles to a **relative** offset byte. This is a signed (two's complement) offset byte which is added to the PC in two's complement arithmetic if the jump is executed.

The range of the jump is therefore −128 to +127 Program Memory bytes relative to the first byte following the instruction.

Jump Instructions

Table 8 shows the list of unconditional jumps.

Table 8. Unconditional Jumps in MCS®-51 Devices

Mnemonic	Operation	Execution Time (μs)
JMP addr	Jump to addr	2
JMP @A+DPTR	Jump to A+DPTR	2
CALL addr	Call subroutine at addr	2
RET	Return from subroutine	2
RETI	Return from interrupt	2
NOP	No operation	1

The Table lists a single "JMP addr" instruction, but in fact there are three—SJMP, LJMP and AJMP—which differ in the format of the destination address. JMP is a generic mnemonic which can be used if the programmer does not care which way the jump is encoded.

The SJMP instruction encodes the destination address as a relative offset, as described above. The instruction is 2 bytes long, consisting of the opcode and the relative offset byte. The jump distance is limited to a range of −128 to +127 bytes relative to the instruction following the SJMP.

The LJMP instruction encodes the destination address as a 16-bit constant. The instruction is 3 bytes long, consisting of the opcode and two address bytes. The destination address can be anywhere in the 64K Program Memory space.

The AJMP instruction encodes the destination address as an 11-bit constant. The instruction is 2 bytes long, consisting of the opcode, which itself contains 3 of the 11 address bits, followed by another byte containing the low 8 bits of the destination address. When the instruction is executed, these 11 bits are simply substituted for the low 11 bits in the PC. The high 5 bits stay the same. Hence the destination has to be within the same 2K block as the instruction following the AJMP.

In all cases the programmer specifies the destination address to the assembler in the same way: as a label or as a 16-bit constant. The assembler will put the destination address into the correct format for the given instruction. If the format required by the instruction will not support the distance to the specified destination address, a "Destination out of range" message is written into the List file.

The JMP @A+DPTR instruction supports case jumps. The destination address is computed at execution time as the sum of the 16-bit DPTR register and the Accumulator. Typically, DPTR is set up with the address of a jump table, and the Accumulator is given an index to the table. In a 5-way branch, for example, an integer 0 through 4 is loaded into the Accumulator. The code to be executed might be as follows:

```
MOV     DPTR,#JUMP_TABLE
MOV     A,INDEX_NUMBER
RL      A
JMP     @A+DPTR
```

The RL A instruction converts the index number (0 through 4) to an even number on the range 0 through 8, because each entry in the jump table is 2 bytes long:

```
JUMP_TABLE:
        AJMP    CASE_0
        AJMP    CASE_1
        AJMP    CASE_2
        AJMP    CASE_3
        AJMP    CASE_4
```

Table 8 shows a single "CALL addr" instruction, but there are two of them—LCALL and ACALL—which differ in the format in which the subroutine address is given to the CPU. CALL is a generic mnemonic which can be used if the programmer does not care which way the address is encoded.

The LCALL instruction uses the 16-bit address format, and the subroutine can be anywhere in the 64K Program Memory space. The ACALL instruction uses the 11-bit format, and the subroutine must be in the same 2K block as the instruction following the ACALL.

In any case the programmer specifies the subroutine address to the assembler in the same way: as a label or as a 16-bit constant. The assembler will put the address into the correct format for the given instructions.

Subroutines should end with a RET instruction, which returns execution to the instruction following the CALL.

RETI is used to return from an interrupt service routine. The only difference between RET and RETI is that RETI tells the interrupt control system that the interrupt in progress is done. If there is no interrupt in progress at the time RETI is executed, then the RETI is functionally identical to RET.

Table 9 shows the list of conditional jumps available to the MCS-51 user. All of these jumps specify the destination address by the relative offset method, and so are limited to a jump distance of −128 to +127 bytes from the instruction following the conditional jump instruction. Important to note, however, the user specifies to the assembler the actual destination address the same way as the other jumps: as a label or a 16-bit constant.

Table 9. Conditional Jumps in MCS®-51 Devices

Mnemonic	Operation	Addressing Modes				Execution Time (μs)
		Dir	Ind	Reg	Imm	
JZ rel	Jump if A = 0	Accumulator only				2
JNZ rel	Jump if A ≠ 0	Accumulator only				2
DJNZ <byte>,rel	Decrement and jump if not zero	X		X		2
CJNE A,<byte>,rel	Jump if A ≠ <byte>	X			X	2
CJNE <byte>,#data,rel	Jump if <byte> ≠ #data		X	X		2

There is no Zero bit in the PSW. The JZ and JNZ instructions test the Accumulator data for that condition.

The DJNZ instruction (Decrement and Jump if Not Zero) is for loop control. To execute a loop N times, load a counter byte with N and terminate the loop with a DJNZ to the beginning of the loop, as shown below for N = 10:

```
        MOV     COUNTER,#10
LOOP:   (begin loop)
          *
          *
          *
        (end loop)
        DJNZ    COUNTER,LOOP
        (continue)
```

The CJNE instruction (Compare and Jump if Not Equal) can also be used for loop control as in Figure 12. Two bytes are specified in the operand field of the instruction. The jump is executed only if the two bytes are not equal. In the example of Figure 12, the two bytes were the data in R1 and the constant 2AH. The initial data in R1 was 2EH. Every time the loop was executed, R1 was decremented, and the looping was to continue until the R1 data reached 2AH.

Another application of this instruction is in "greater than, less than" comparisons. The two bytes in the operand field are taken as unsigned integers. If the first is less than the second, then the Carry bit is set (1). If the first is greater than or equal to the second, then the Carry bit is cleared.

CPU TIMING

All MCS-51 microcontrollers have an on-chip oscillator which can be used if desired as the clock source for the CPU. To use the on-chip oscillator, connect a crystal or ceramic resonator between the XTAL1 and XTAL2 pins of the microcontroller, and capacitors to ground as shown in Figure 13.

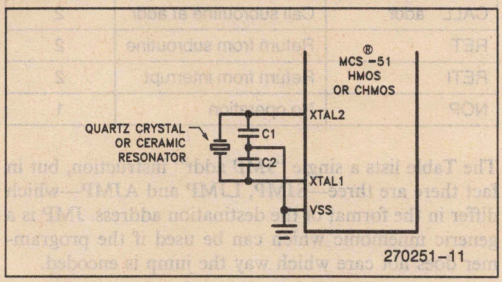


Figure 13. Using the On-Chip Oscillator

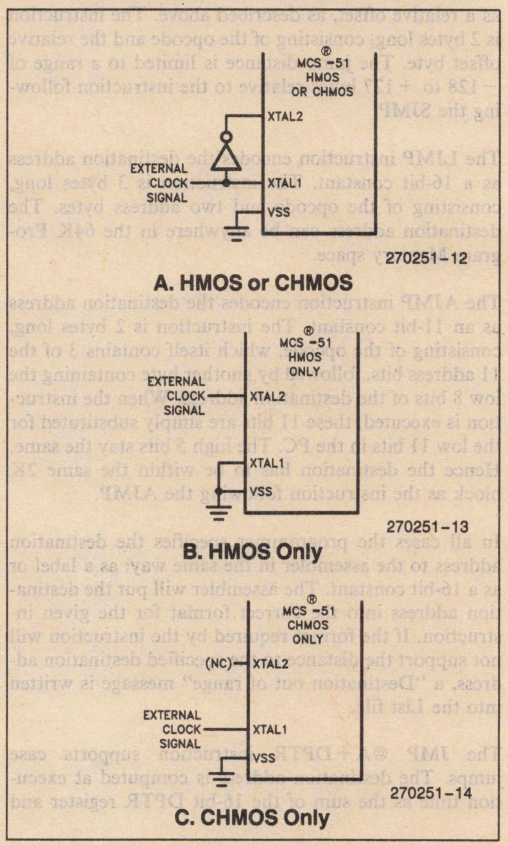


Figure 14. Using an External Clock

Examples of how to drive the clock with an external oscillator are shown in Figure 14. Note that in the HMOS devices (8051, etc.) the signal at the XTAL2 pin actually drives the internal clock generator. In the CHMOS devices (80C51BH, etc.) the signal at the XTAL1 pin drives the internal clock generator. If only one pin is going to be driven with the external oscillator signal, make sure it is the right pin.

The internal clock generator defines the sequence of states that make up the MCS-51 machine cycle.

Machine Cycles

A machine cycle consists of a sequence of 6 states, numbered S1 through S6. Each state time lasts for two oscillator periods. Thus a machine cycle takes 12 oscillator periods or 1 μs if the oscillator frequency is 12 MHz.

Each state is divided into a Phase 1 half and a Phase 2 half. Figure 15 shows the fetch/execute sequences in

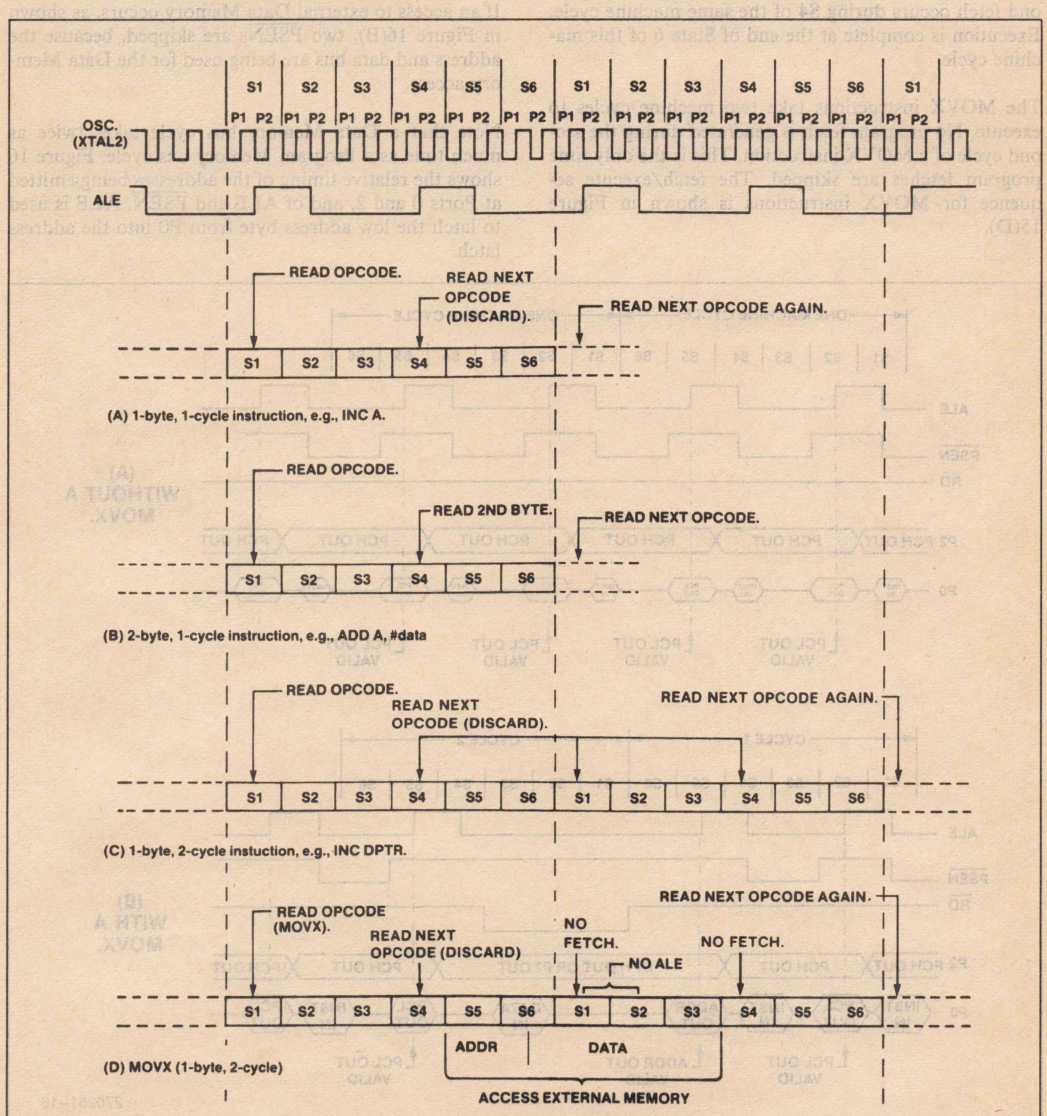


Figure 15. State Sequences in MCS®-51 Devices

states and phases for various kinds of instructions. Normally two program fetches are generated during each machine cycle, even if the instruction being executed doesn't require it. If the instruction being executed doesn't need more code bytes, the CPU simply ignores the extra fetch, and the Program Counter is not incremented.

Execution of a one-cycle instruction (Figure 15A and B) begins during State 1 of the machine cycle, when the opcode is latched into the Instruction Register. A second fetch occurs during S4 of the same machine cycle. Execution is complete at the end of State 6 of this machine cycle.

The MOVX instructions take two machine cycles to execute. No program fetch is generated during the second cycle of a MOVX instruction. This is the only time program fetches are skipped. The fetch/execute sequence for MOVX instructions is shown in Figure 15(D).

The fetch/execute sequences are the same whether the Program Memory is internal or external to the chip. Execution times do not depend on whether the Program Memory is internal or external.

Figure 16 shows the signals and timing involved in program fetches when the Program Memory is external. If Program Memory is external, then the Program Memory read strobe $\overline{PSEN}$ is normally activated twice per machine cycle, as shown in Figure 16(A).

If an access to external Data Memory occurs, as shown in Figure 16(B), two $\overline{PSEN}$s are skipped, because the address and data bus are being used for the Data Memory access.

Note that a Data Memory bus cycle takes twice as much time as a Program Memory bus cycle. Figure 16 shows the relative timing of the addresses being emitted at Ports 0 and 2, and of ALE and $\overline{PSEN}$. ALE is used to latch the low address byte from P0 into the address latch.

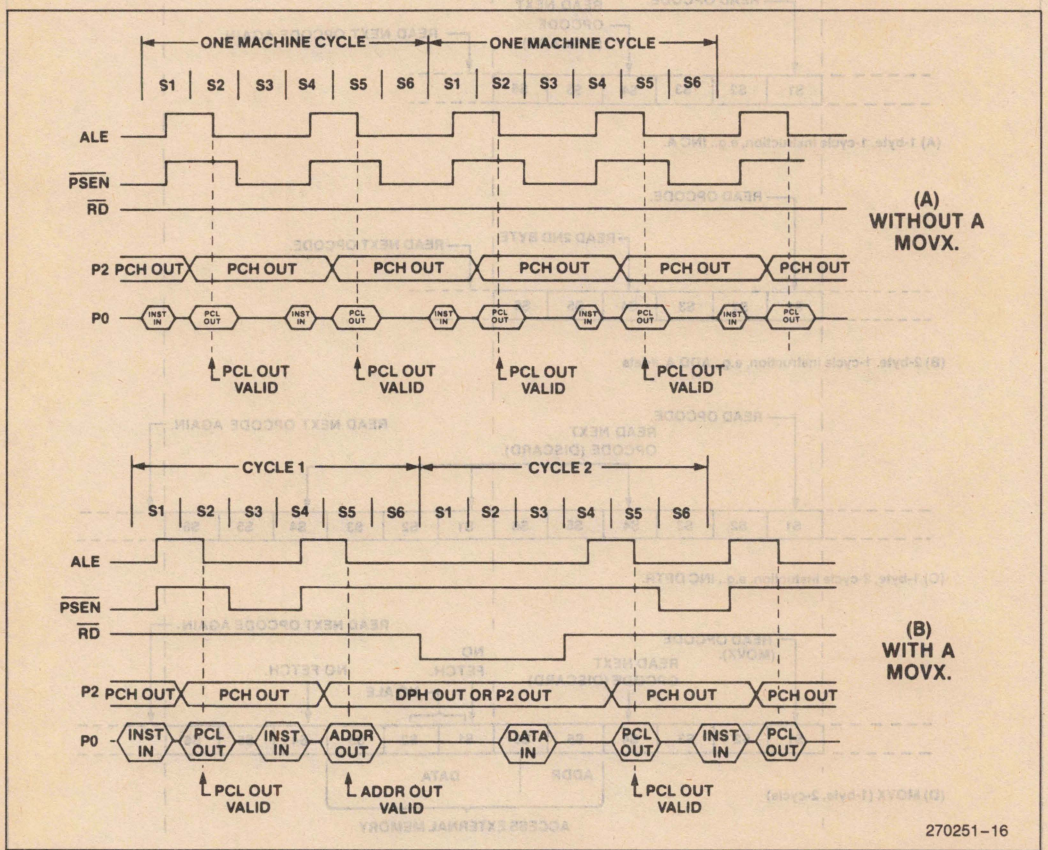


Figure 16. Bus Cycles in MCS®-51 Devices Executing from External Program Memory

When the CPU is executing from internal Program Memory, PSEN is not activated, and program addresses are not emitted. However, ALE continues to be activated twice per machine cycle and so is available as a clock output signal. Note, however, that one ALE is skipped during the execution of the MOVX instruction.

Interrupt Structure

The 8051, 8051AH, and 80C51BH, and their ROMless and EPROM versions, provide 5 interrupt sources: 2 external interrupts, 2 timer interrupts, and the serial port interrupt. The 8052AH provides these 5 plus a sixth interrupt that is associated with the third timer/counter which is present in this device.

What follows is an overview of the interrupt structure for these devices. More detailed information for specific members of the MCS-51 family is provided in the chapters of this handbook that describe the specific devices.

INTERRUPT ENABLES

Each of the interrupt sources can be individually enabled or disabled by setting or clearing a bit in the SFR named IE (Interrupt Enable). This register also contains a global disable bit, which can be cleared to disable all interrupts at once. Figure 17 shows the IE register for the 8052AH.

(MSB)							(LSB)
EA	X	ET2	ES	ET1	EX1	ET0	EX0

Symbol	Position	Function
EA	IE.7	disables all interrupts. If EA = 0, no interrupt will be acknowledged. If EA = 1, each interrupt source is individually enabled or disabled by setting or clearing its enable bit.
—	IE.6	reserved
ET2	IE.5	enables or disables the Timer 2 overflow or capture interrupt. If ET2 = 0, the Timer 2 interrupt is disabled.
ES	IE.4	enables or disables the Serial Port interrupt. If ES = 0, the Serial Port interrupt is disabled.
ET1	IE.3	enables or disables the Timer 1 Overflow interrupt. If ET1 = 0, the Timer 1 interrupt is disabled.
EX1	IE.2	enables or disables External Interrupt 1. If EX1 = 0, External Interrupt 1 is disabled.
ET0	IE.1	enables or disables the Timer 0 Overflow interrupt. If ET0 = 0, the Timer 0 interrupt is disabled.
EX0	IE.0	enables or disables External Interrupt 0. If EX0 = 0, External Interrupt 0 is disabled.

Figure 17. IE (Interrupt Enable) Register in the 8052AH

(MSB)							(LSB)
X	X	PT2	PS	PT1	PX1	PT0	PX0

Symbol	Position	Function
—	IP.7	reserved
—	IP.6	reserved
PT2	IP.5	defines the Timer 2 interrupt priority level. PT2 = 1 programs it to the higher priority level.
PS	IP.4	defines the Serial Port interrupt priority level. PS = 1 programs it to the higher priority level.
PT1	IP.3	defines the Timer 1 interrupt priority level. PT1 = 1 programs it to the higher priority level.
PX1	IP.2	defines the External Interrupt 1 priority level. PX1 = 1 programs it to the higher priority level.
PT0	IP.1	defines the Timer 0 interrupt priority level. PT0 = 1 programs it to the higher priority level.
PX0	IP.0	defines the External Interrupt 0 priority level. PX0 = 1 programs it to the higher priority level.

Figure 18. IP (Interrupt Priority) Register in the 8052AH

INTERRUPT PRIORITIES

Each interrupt source can also be individually programmed to one of two priority levels by setting or clearing a bit in the SFR named IP (Interrupt Priority). Figure 18 shows the IP register in the 8052AH.

A low-priority interrrupt can be interrupted by a high-priority interrupt, but not by another low-priority interrupt. A high-priority interrupt can't be interrupted by any other interrupt source.

If two interrupt requests of different priority levels are received simultaneously, the request of higher priority level is serviced. If interrupt requests of the same priority level are received simultaneously, an internal polling sequence determines which request is serviced. Thus within each priority level there is a second priority structure determined by the polling sequence.

Figure 19 shows, for the 8052AH, how the IE and IP registers and the polling sequence work to determine which if any interrupt will be serviced.

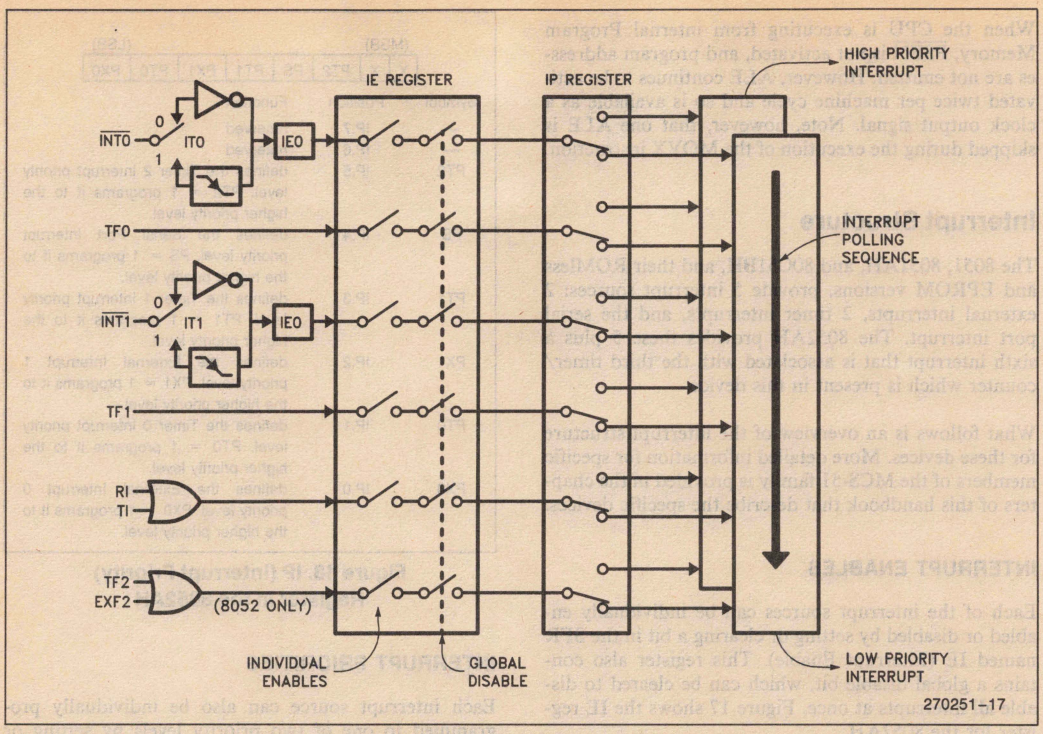


Figure 19. 8052 Interrupt Control System

In operation, all the interrupt flags are latched into the interrupt control system during State 5 of every machine cycle. The samples are polled during the following machine cycle. If the flag for an enabled interrupt is found to be set (1), the interrupt system generates an LCALL to the appropriate location in Program Memory, unless some other condition blocks the interrupt. Several conditions can block an interrupt, among them that an interrupt of equal or higher priority level is already in progress.

The hardware-generated LCALL causes the contents of the Program Counter to be pushed onto the stack, and reloads the PC with the beginning address of the service routine. As previously noted (Figure 3), the service routine for each interrupt begins at a fixed location.

Only the Program Counter is automatically pushed onto the stack, not the PSW or any other register. Having only the PC be automatically saved allows the programmer to decide how much time to spend saving which other registers. This enhances the interrupt response time, albeit at the expense of increasing the programmer's burden of responsibility. As a result, many interrupt functions that are typical in control applications—toggling a port pin, for example, or reloading a timer, or unloading a serial buffer—can often be completed in less time than it takes other architectures to commence them.

SIMULATING A THIRD PRIORITY LEVEL IN SOFTWARE

Some applications require more than the two priority levels that are provided by on-chip hardware in MCS-51 devices. In these cases, relatively simple software can be written to produce the same effect as a third priority level.

First, interrupts that are to have higher priority than 1 are assigned to priority 1 in the IP (Interrupt Priority) register. The service routines for priority 1 interrupts that are supposed to be interruptible by "priority 2" interrupts are written to include the following code:

```
        PUSH    IE
        MOV     IE,#MASK
        CALL    LABEL
          *******
  (execute service routine)
          *******
        POP     IE
        RET
LABEL:  RETI
```

As soon as any priority 1 interrupt is acknowledged, the IE (Interrupt Enable) register is re-defined so as to disable all but "priority 2" interrupts. Then, a CALL to LABEL executes the RETI instruction, which clears the priority 1 interrupt-in-progress flip-flop. At this point any priority 1 interrupt that is enabled can be serviced, but only "priority 2" interrupts are enabled.

POPping IE restores the original enable byte. Then a normal RET (rather than another RETI) is used to terminate the service routine. The additional software adds 10 μs (at 12 MHz) to priority 1 interrupts.

Hardware Description of the 8051 and 8052

6

6 Hardware Description of the 8051 and 8052

HARDWARE DESCRIPTION OF THE 8051 AND 8052

INTRODUCTION

This chapter presents a comprehensive description of the on-chip hardware features of the MCS®-51 microcontrollers. Included in this description are

- The port drivers and how they function both as ports and, for Ports 0 and 2, in bus operations
- The Timer/Counters
- The Serial Interface
- The Interrupt System
- Reset
- The Reduced Power Modes in the CHMOS devices
- The EPROM versions of the 8051AH, 8052AH, and 80C51BH

The devices under consideration are listed in Table 1. As it becomes unwieldy to be constantly referring to each of these devices by their individual names, we will adopt a convention of referring to them generically as 8051s and 8052s, unless a specific member of the group is being referred to, in which case it will be specifically named. The "8051s" include the 8051, 8051AH, and 80C51BH, and their ROMless and EPROM versions. The "8052s" are the 8052AH, 8032AH, and 8752BH.

Figure 1 shows a functional block diagram of the 8051s and 8052s.

Table 1. The MCS-51 Family of Microcontrollers

Device Name	ROMless Version	EPROM Version	ROM Bytes	RAM Bytes	16-bit Timers	Ckt Type
8051	8031	(8751)	4K	128	2	HMOS
8051AH	8031AH	8751H	4K	128	2	HMOS
8052AH	8032AH	8752BH	8K	256	3	HMOS
80C51BH	80C31BH	87C51	4K	128	2	CHMOS

Special Function Registers

A map of the on-chip memory area called SFR (Special Function Register) space is shown in Figure 2. SFRs marked by parentheses are resident in the 8052s but not in the 8051s.

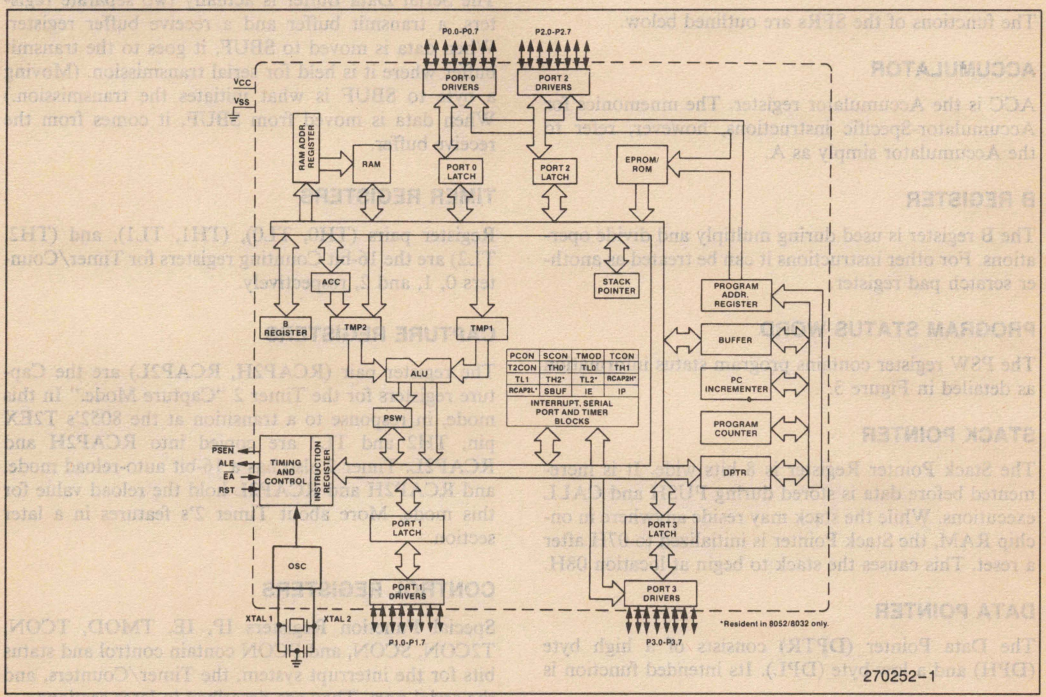


Figure 1. MCS-51 Architectural Block Diagram

	8 Bytes							
F8								FF
F0	B							F7
E8								EF
E0	ACC							E7
D8								DF
D0	PSW							D7
C8	(T2CON)		(RCAP2L)	(RCAP2H)	(TL2)	(TH2)		CF
C0								C7
B8	IP							BF
B0	P3							B7
A8	IE							AF
A0	P2							A7
98	SCON	SBUF						9F
90	P1							97
88	TCON	TMOD	TL0	TL1	TH0	TH1		8F
80	P0	SP	DPL	DPH			PCON	87

Figure 2. SFR Map. (. . .) Indicates Resident in 8052s, not in 8051s

Note that not all of the addresses are occupied. Unoccupied addresses are not implemented on the chip. Read accesses to these addresses will in general return random data, and write accesses will have no effect.

User software should not write 1s to these unimplemented locations, since they may be used in future MCS-51 products to invoke new features. In that case the reset or inactive values of the new bits will always be 0, and their active values will be 1.

The functions of the SFRs are outlined below.

ACCUMULATOR

ACC is the Accumulator register. The mnemonics for Accumulator-Specific instructions, however, refer to the Accumulator simply as A.

B REGISTER

The B register is used during multiply and divide operations. For other instructions it can be treated as another scratch pad register.

PROGRAM STATUS WORD

The PSW register contains program status information as detailed in Figure 3.

STACK POINTER

The Stack Pointer Register is 8 bits wide. It is incremented before data is stored during PUSH and CALL executions. While the stack may reside anywhere in on-chip RAM, the Stack Pointer is initialized to 07H after a reset. This causes the stack to begin at location 08H.

DATA POINTER

The Data Pointer (DPTR) consists of a high byte (DPH) and a low byte (DPL). Its intended function is to hold a 16-bit address. It may be manipulated as a 16-bit register or as two independent 8-bit registers.

PORTS 0 TO 3

P0, P1, P2 and P3 are the SFR latches of Ports 0, 1, 2 and 3, respectively.

SERIAL DATA BUFFER

The Serial Data Buffer is actually two separate registers, a transmit buffer and a receive buffer register. When data is moved to SBUF, it goes to the transmit buffer where it is held for serial transmission. (Moving a byte to SBUF is what initiates the transmission.) When data is moved from SBUF, it comes from the receive buffer.

TIMER REGISTERS

Register pairs (TH0, TL0), (TH1, TL1), and (TH2, TL2) are the 16-bit Counting registers for Timer/Counters 0, 1, and 2, respectively.

CAPTURE REGISTERS

The register pair (RCAP2H, RCAP2L) are the Capture registers for the Timer 2 "Capture Mode." In this mode, in response to a transition at the 8052's T2EX pin, TH2 and TL2 are copied into RCAP2H and RCAP2L. Timer 2 also has a 16-bit auto-reload mode, and RCAP2H and RCAP2L hold the reload value for this mode. More about Timer 2's features in a later section.

CONTROL REGISTERS

Special Function Registers IP, IE, TMOD, TCON, T2CON, SCON, and PCON contain control and status bits for the interrupt system, the Timer/Counters, and the serial port. They are described in later sections.

(MSB)							(LSB)
CY	AC	F0	RS1	RS0	OV	—	P

Symbol	Position	Name and Significance
CY	PSW.7	Carry flag.
AC	PSW.6	Auxiliary Carry flag. (For BCD operations.)
F0	PSW.5	Flag 0 (Available to the user for general purposes.)
RS1	PSW.4	Register bank select control bits 1 &
RS0	PSW.3	0. Set/cleared by software to determine working register bank (see Note).
OV	PSW.2	Overflow flag.
—	PSW.1	User definable flag.
P	PSW.0	Parity flag. Set/cleared by hardware each instruction cycle to indicate an odd/ even number of "one" bits in the Accumulator, i.e., even parity.

NOTE:
The contents of (RS1, RS0) enable the working register banks as follows:

(0.0)—Bank 0 (00H–07H)
(0.1)—Bank 1 (08H–0FH)
(1.0)—Bank 2 (10H–17H)
(1.1)—Bank 3 (18H–1FH)

Figure 3. PSW: Program Status Word Register

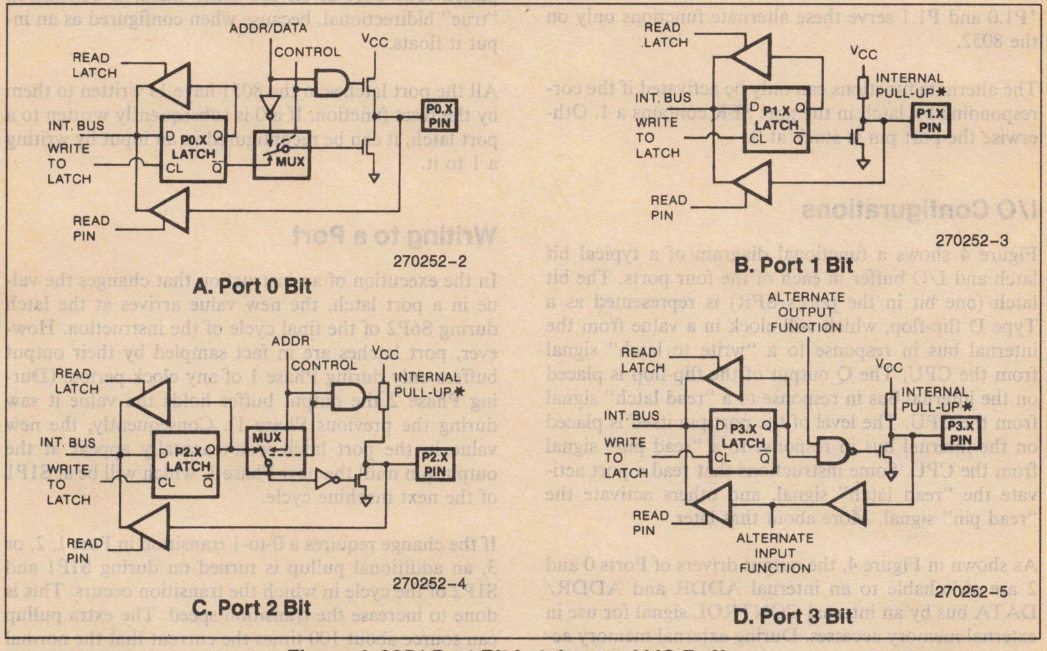


Figure 4. 8051 Port Bit Latches and I/O Buffers
*See Figure 5 for details of the internal pullup.

PORT STRUCTURES AND OPERATION

All four ports in the 8051 are bidirectional. Each consists of a latch (Special Function Registers P0 through P3), an output driver, and an input buffer.

The output drivers of Ports 0 and 2, and the input buffers of Port 0, are used in accesses to external memory. In this application, Port 0 outputs the low byte of the external memory address, time-multiplexed with the byte being written or read. Port 2 outputs the high byte of the external memory address when the address is 16 bits wide. Otherwise the Port 2 pins continue to emit the P2 SFR content.

All the Port 3 pins, and (in the 8052) two Port 1 pins are multifunctional. They are not only port pins, but also serve the functions of various special features as listed on the following page.

Port Pin	Alternate Function
*P1.0	T2 (Timer/Counter 2 external input)
*P1.1	T2EX (Timer/Counter 2 Capture/Reload trigger)
P3.0	RXD (serial input port)
P3.1	TXD (serial output port)
P3.2	$\overline{INT0}$ (external interrupt)
P3.3	$\overline{INT1}$ (external interrupt)
P3.4	T0 (Timer/Counter 0 external input)
P3.5	T1 (Timer/Counter 1 external input)
P3.6	$\overline{WR}$ (external Data Memory write strobe)
P3.7	$\overline{RD}$ (external Data Memory read strobe)

*P1.0 and P1.1 serve these alternate functions only on the 8052.

The alternate functions can only be activated if the corresponding bit latch in the port SFR contains a 1. Otherwise the port pin is stuck at 0.

I/O Configurations

Figure 4 shows a functional diagram of a typical bit latch and I/O buffer in each of the four ports. The bit latch (one bit in the port's SFR) is represented as a Type D flip-flop, which will clock in a value from the internal bus in response to a "write to latch" signal from the CPU. The Q output of the flip-flop is placed on the internal bus in response to a "read latch" signal from the CPU. The level of the port pin itself is placed on the internal bus in response to a "read pin" signal from the CPU. Some instructions that read a port activate the "read latch" signal, and others activate the "read pin" signal. More about that later.

As shown in Figure 4, the output drivers of Ports 0 and 2 are switchable to an internal ADDR and ADDR/DATA bus by an internal CONTROL signal for use in external memory accesses. During external memory accesses, the P2 SFR remains unchanged, but the P0 SFR gets 1s written to it.

Also shown in Figure 4, is that if a P3 bit latch contains a 1, then the output level is controlled by the signal labeled "alternate output function." The actual P3.X pin level is always available to the pin's alternate input function, if any.

Ports 1, 2, and 3 have internal pullups. Port 0 has open drain outputs. Each I/O line can be independently used as an input or an output. (Ports 0 and 2 may not be used as general purpose I/O when being used as the ADDR/DATA BUS). To be used as an input, the port bit latch must contain a 1, which turns off the output driver FET. Then, for Ports 1, 2, and 3, the pin is pulled high by the internal pullup, but can be pulled low by an external source.

Port 0 differs in not having internal pullups. The pullup FET in the P0 output driver (see Figure 4) is used only when the Port is emitting 1s during external memory accesses. Otherwise the pullup FET is off. Consequently P0 lines that are being used as output port lines are open drain. Writing a 1 to the bit latch leaves both output FETs off, so the pin floats. In that condition it can be used a high-impedance input.

Because Ports 1, 2, and 3 have fixed internal pullups they are sometimes called "quasi-bidirectional" ports. When configured as inputs they pull high and will source current (I_{IL}, in the data sheets) when externally pulled low. Port 0, on the other hand, is considered "true" bidirectional, because when configured as an input it floats.

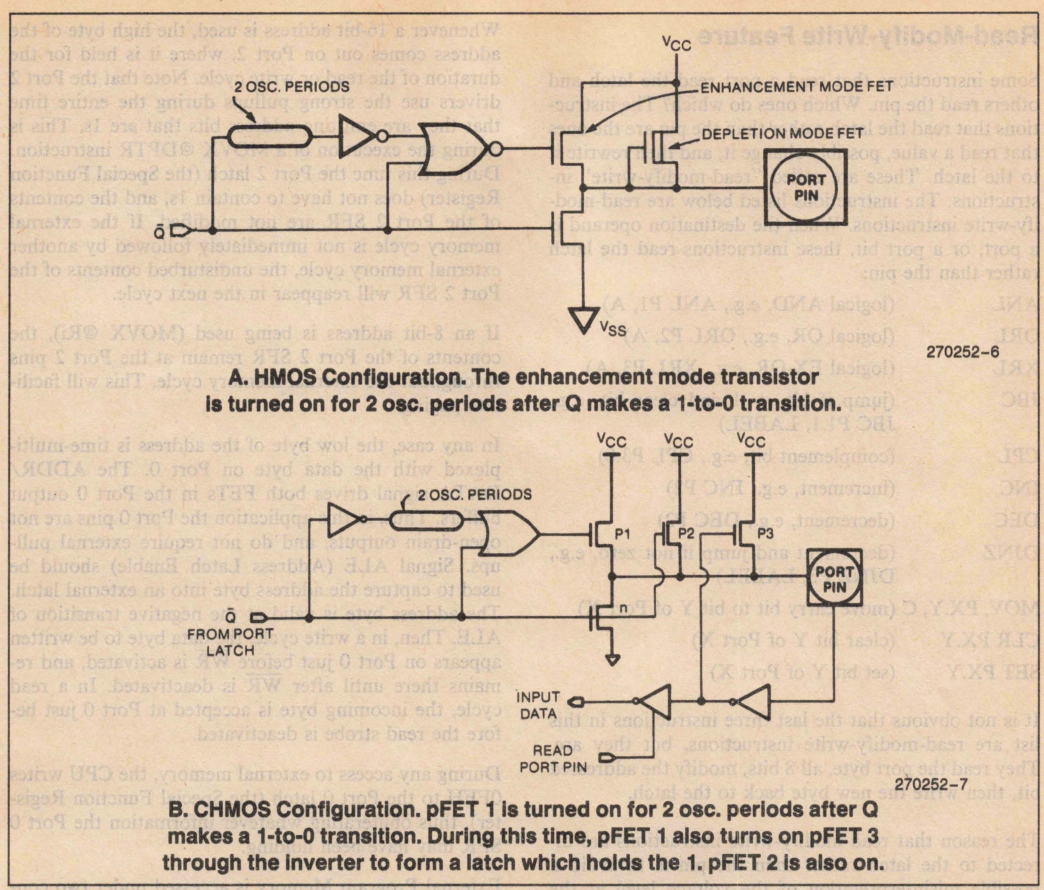
All the port latches in the 8051 have 1s written to them by the reset function. If a 0 is subsequently written to a port latch, it can be reconfigured as an input by writing a 1 to it.

Writing to a Port

In the execution of an instruction that changes the value in a port latch, the new value arrives at the latch during S6P2 of the final cycle of the instruction. However, port latches are in fact sampled by their output buffers only during Phase 1 of any clock period. (During Phase 2 the output buffer holds the value it saw during the previous Phase 1). Consequently, the new value in the port latch won't actually appear at the output pin until the next Phase 1, which will be at S1P1 of the next machine cycle.

If the change requires a 0-to-1 transition in Port 1, 2, or 3, an additional pullup is turned on during S1P1 and S1P2 of the cycle in which the transition occurs. This is done to increase the transition speed. The extra pullup can source about 100 times the current that the normal pullup can. It should be noted that the internal pullups are field-effect transistors, not linear resistors. The pullup arrangements are shown in Figure 5.

In HMOS versions of the 8051, the fixed part of the pullup is a depletion-mode transistor with the gate wired to the source. This transistor will allow the pin to source about 0.25 mA when shorted to ground. In parallel with the fixed pullup is an enhancement-mode transistor, which is activated during S1 whenever the port bit does a 0-to-1 transition. During this interval, if the port pin is shorted to ground, this extra transistor will allow the pin to source an additional 30 mA.



A. HMOS Configuration. The enhancement mode transistor is turned on for 2 osc. periods after Q makes a 1-to-0 transition.

B. CHMOS Configuration. pFET 1 is turned on for 2 osc. periods after Q makes a 1-to-0 transition. During this time, pFET 1 also turns on pFET 3 through the inverter to form a latch which holds the 1. pFET 2 is also on.

Figure 5. Ports 1 And 3 HMOS And CHMOS Internal Pullup Configurations. Port 2 Is Similar Except That It Holds The Strong Pullup On While Emitting 1s That Are Address Bits. (See Text, "Accessing External Memory".)

In the CHMOS versions, the pullup consists of three pFETs. It should be noted that an n-channel FET (nFET) is turned on when a logical 1 is applied to its gate, and is turned off when a logical 0 is applied to its gate. A p-channel FET (pFET) is the opposite: it is on when its gate sees a 0, and off when its gate sees a 1.

pFET1 in Figure 5 is the transistor that is turned on for 2 oscillator periods after a 0-to-1 transition in the port latch. While it's on, it turns on pFET3 (a weak pull-up), through the inverter. This inverter and pFET form a latch which hold the 1.

Note that if the pin is emitting a 1, a negative glitch on the pin from some external source can turn off pFET3, causing the pin to go into a float state. pFET2 is a very weak pullup which is on whenever the nFET is off, in traditional CMOS style. It's only about $1/10$ the strength of pFET3. Its function is to restore a 1 to the pin in the event the pin *had* a 1 and lost it to a glitch.

Port Loading and Interfacing

The output buffers of Ports 1, 2, and 3 can each drive 4 LS TTL inputs. These ports on HMOS versions can be driven in a normal manner by any TTL or NMOS circuit. Both HMOS and CHMOS pins can be driven by open-collector and open-drain outputs, but note that 0-to-1 transitions will not be fast. In the HMOS device, if the pin is driven by an open-collector output, a 0-to-1 transition will have to be driven by the relatively weak depletion mode FET in Figure 5(A). In the CHMOS device, an input 0 turns off pullup pFET3, leaving only the very weak pullup pFET2 to drive the transition.

Port 0 output buffers can each drive 8 LS TTL inputs. They do, however, require external pullups to drive NMOS inputs, except when being used as the ADDRESS/DATA bus.

Read-Modify-Write Feature

Some instructions that read a port read the latch and others read the pin. Which ones do which? The instructions that read the latch rather than the pin are the ones that read a value, possibly change it, and then rewrite it to the latch. These are called "read-modify-write" instructions. The instructions listed below are read-modify-write instructions. When the destination operand is a port, or a port bit, these instructions read the latch rather than the pin:

ANL	(logical AND, e.g., ANL P1, A)
ORL	(logical OR, e.g., ORL P2, A)
XRL	(logical EX-OR, e.g., XRL P3, A)
JBC	(jump if bit = 1 and clear bit, e.g., JBC P1.1, LABEL)
CPL	(complement bit, e.g., CPL P3.0)
INC	(increment, e.g., INC P2)
DEC	(decrement, e.g., DEC P2)
DJNZ	(decrement and jump if not zero, e.g., DJNZ P3, LABEL)
MOV, PX.Y, C	(move carry bit to bit Y of Port X)
CLR PX.Y	(clear bit Y of Port X)
SET PX.Y	(set bit Y of Port X)

It is not obvious that the last three instructions in this list are read-modify-write instructions, but they are. They read the port byte, all 8 bits, modify the addressed bit, then write the new byte back to the latch.

The reason that read-modify-write instructions are directed to the latch rather than the pin is to avoid a possible misinterpretation of the voltage level at the pin. For example, a port bit might be used to drive the base of a transistor. When a 1 is written to the bit, the transistor is turned on. If the CPU then reads the same port bit at the pin rather than the latch, it will read the base voltage of the transistor and interpret it as a 0. Reading the latch rather than the pin will return the correct value of 1.

ACCESSING EXTERNAL MEMORY

Accesses to external memory are of two types: accesses to external Program Memory and accesses to external Data Memory. Accesses to external Program Memory use signal $\overline{PSEN}$ (program store enable) as the read strobe. Accesses to external Data Memory use $\overline{RD}$ or $\overline{WR}$ (alternate functions of P3.7 and P3.6) to strobe the memory.

Fetches from external Program Memory always use a 16-bit address. Accesses to external Data Memory can use either a 16-bit address (MOVX @DPTR) or an 8-bit address (MOVX @Ri).

Whenever a 16-bit address is used, the high byte of the address comes out on Port 2, where it is held for the duration of the read or write cycle. Note that the Port 2 drivers use the strong pullups during the entire time that they are emitting address bits that are 1s. This is during the execution of a MOVX @DPTR instruction. During this time the Port 2 latch (the Special Function Register) does not have to contain 1s, and the contents of the Port 2 SFR are not modified. If the external memory cycle is not immediately followed by another external memory cycle, the undisturbed contents of the Port 2 SFR will reappear in the next cycle.

If an 8-bit address is being used (MOVX @Ri), the contents of the Port 2 SFR remain at the Port 2 pins throughout the external memory cycle. This will facilitate paging.

In any case, the low byte of the address is time-multiplexed with the data byte on Port 0. The ADDR/DATA signal drives both FETs in the Port 0 output buffers. Thus, in this application the Port 0 pins are not open-drain outputs, and do not require external pullups. Signal ALE (Address Latch Enable) should be used to capture the address byte into an external latch. The address byte is valid at the negative transition of ALE. Then, in a write cycle, the data byte to be written appears on Port 0 just before $\overline{WR}$ is activated, and remains there until after $\overline{WR}$ is deactivated. In a read cycle, the incoming byte is accepted at Port 0 just before the read strobe is deactivated.

During any access to external memory, the CPU writes 0FFH to the Port 0 latch (the Special Function Register), thus obliterating whatever information the Port 0 SFR may have been holding.

External Program Memory is accessed under two conditions:

1) Whenever signal $\overline{EA}$ is active; or
2) Whenever the program counter (PC) contains a number that is larger than 0FFFH (1FFFH for the 8052).

This requires that the ROMless versions have $\overline{EA}$ wired low to enable the lower 4K (8K for the 8032) program bytes to be fetched from external memory.

When the CPU is executing out of external Program Memory, all 8 bits of Port 2 are dedicated to an output function and may not be used for general purpose I/O. During external program fetches they output the high byte of the PC. During this time the Port 2 drivers use the strong pullups to emit PC bits that are 1s.

TIMER/COUNTERS

The 8051 has two 16-bit Timer/Counter registers: Timer 0 and Timer 1. The 8052 has these two plus one more: Timer 2. All three can be configured to operate either as timers or event counters.

In the "Timer" function, the register is incremented every machine cycle. Thus, one can think of it as counting machine cycles. Since a machine cycle consists of 12 oscillator periods, the count rate is $1/12$ of the oscillator frequency.

In the "Counter" function, the register is incremented in response to a 1-to-0 transition at its corresponding external input pin, T0, T1 or (in the 8052) T2. In this function, the external input is sampled during S5P2 of every machine cycle. When the samples show a high in one cycle and a low in the next cycle, the count is incremented. The new count value appears in the register during S3P1 of the cycle following the one in which the transition was detected. Since it takes 2 machine cycles (24 oscillator periods) to recognize a 1-to-0 transition, the maximum count rate is $1/24$ of the oscillator frequency. There are no restrictions on the duty cycle of the external input signal, but to ensure that a given level is sampled at least once before it changes, it should be held for at least one full machine cycle.

In addition to the "Timer" or "Counter" selection, Timer 0 and Timer 1 have four operating modes from which to select. Timer 2, in the 8052, has three modes of operation: "Capture," "Auto-Reload" and "baud rate generator."

Timer 0 and Timer 1

These Timer/Counters are present in both the 8051 and the 8052. The "Timer" or "Counter" function is selected by control bits C/$\overline{T}$ in the Special Function Register TMOD (Figure 6). These two Timer/Counters have four operating modes, which are selected by bit-pairs (M1, M0) in TMOD. Modes 0, 1, and 2 are the same for both Timer/Counters. Mode 3 is different. The four operating modes are described in the following text.

MODE 0

Putting either Timer into Mode 0 makes it look like an 8048 Timer, which is an 8-bit Counter with a divide-by-32 prescaler. Figure 7 shows the Mode 0 operation as it applies to Timer 1.

In this mode, the Timer register is configured as a 13-Bit register. As the count rolls over from all 1s to all 0s, it sets the Timer interrupt flag TF1. The counted input is enabled to the Timer when TR1 = 1 and either GATE = 0 or $\overline{INT1}$ = 1. (Setting GATE = 1 allows the Timer to be controlled by external input $\overline{INT1}$, to facilitate pulse width measurements.) TR1 is a control bit in the Special Function Register TCON (Figure 8). GATE is in TMOD.

The 13-Bit register consists of all 8 bits of TH1 and the lower 5 bits of TL1. The upper 3 bits of TL1 are indeterminate and should be ignored. Setting the run flag (TR1) does not clear the registers.

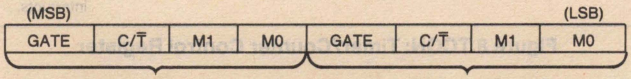
Mode 0 operation is the same for Timer 0 as for Timer 1. Substitute TR0, TF0 and $\overline{INT0}$ for the corresponding Timer 1 signals in Figure 7. There are two different GATE bits, one for Timer 1 (TMOD.7) and one for Timer 0 (TMOD.3).

MODE 1

Mode 1 is the same as Mode 0, except that the Timer register is being run with all 16 bits.

MODE 2

Mode 2 configures the Timer register as an 8-bit Counter (TL1) with automatic reload, as shown in Figure 9. Overflow from TL1 not only sets TF1, but also reloads



GATE	Gating control when set. Timer/Counter "x" is enabled only while "$\overline{INTx}$" pin is high and "TRx" control pin is set. When cleared Timer "x" is enabled whenever "TRx" control bit is set.
C/$\overline{T}$	Timer or Counter Selector cleared for Timer operation (input from internal system clock). Set for Counter operation (input from "Tx" input pin).

M1	M0	Operating Mode
0	0	MCS-48 Timer "TLx" serves as 5-bit prescaler.
0	1	16-bit Timer/Counter "THx" and "TLx" are cascaded; there is no prescaler.
1	0	8-bit auto-reload Timer/Counter "THx" holds a value which is to be reloaded into "TLx" each time it overflows.
1	1	(Timer 0) TL0 is an 8-bit Timer/Counter controlled by the standard Timer 0 control bits. TH0 is an 8-bit timer only controlled by Timer 1 control bits.
1	1	(Timer 1) Timer/Counter 1 stopped.

Figure 6. TMOD: Timer/Counter Mode Control Register

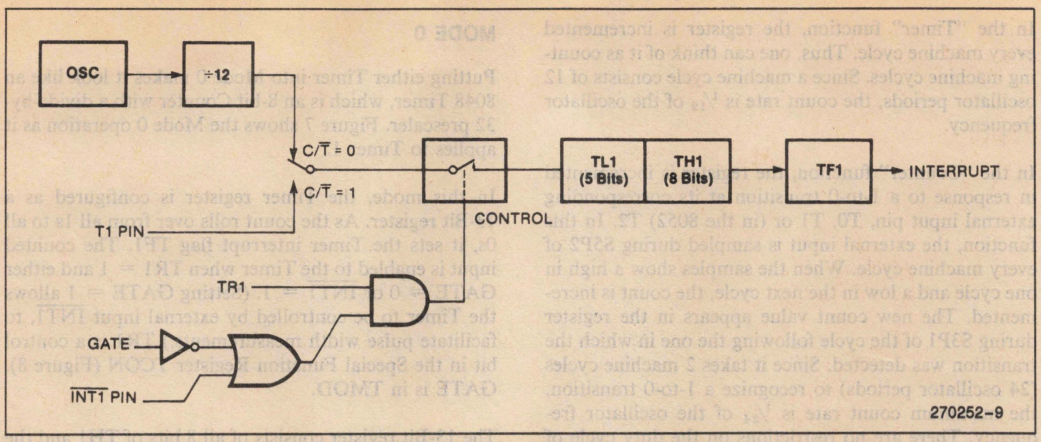


Figure 7. Timer/Counter 1 Mode 0: 13-Bit Counter

(MSB)							(LSB)
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

Symbol	Position	Name and Significance
TF1	TCON.7	Timer 1 overflow Flag. Set by hardware on Timer/Counter overflow. Cleared by hardware when processor vectors to interrupt routine.
TR1	TCON.6	Timer 1 Run control bit. Set/cleared by software to turn Timer/Counter on/off.
TF0	TCON.5	Timer 0 overflow Flag. Set by hardware on Timer/Counter overflow. Cleared by hardware when processor vectors to interrupt routine.
TR0	TCON.4	Timer 0 Run control bit. Set/cleared by software to turn Timer/Counter on/off.
IE1	TCON.3	Interrupt 1 Edge flag. Set by hardware when external interrupt edge detected. Cleared when interrupt processed.
IT1	TCON.2	Interrupt 1 Type control bit. Set/cleared by software to specify falling edge/low level triggered external interrupts.
IE0	TCON.1	Interrupt 0 Edge flag. Set by hardware when external interrupt edge detected. Cleared when interrupt processed.
IT0	TCON.0	Interrupt 0 Type control bit. Set/cleared by software to specify falling edge/low level triggered external interrupts.

Figure 8.TCON: Timer/Counter Control Register

TL1 with the contents of TH1, which is preset by software. The reload leaves TH1 unchanged.

Mode 2 operation is the same for Timer/Counter 0.

MODE 3

Timer 1 in Mode 3 simply holds its count. The effect is the same as setting TR1 = 0.

Timer 0 in Mode 3 establishes TL0 and TH0 as two separate counters. The logic for Mode 3 on Timer 0 is shown in Figure 10. TL0 uses the Timer 0 control bits: C/$\overline{T}$, GATE, TR0, $\overline{INT0}$, and TF0. TH0 is locked into a timer function (counting machine cycles) and takes over the use of TR1 and TF1 from Timer 1. Thus, TH0 now controls the "Timer 1" interrupt.

Mode 3 is provided for applications requiring an extra 8-bit timer or counter. With Timer 0 in Mode 3, an 8051 can look like it has three Timer/Counters, and an 8052, like it has four. When Timer 0 is in Mode 3, Timer 1 can be turned on and off by switching it out of and into its own Mode 3, or can still be used by the serial port as a baud rate generator, or in fact, in any application not requiring an interrupt.

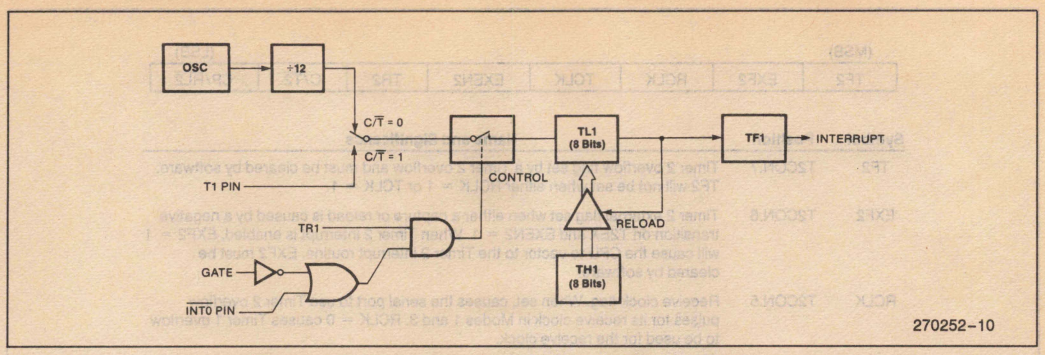


Figure 9. Timer/Counter 1 Mode 2: 8-Bit Auto-Reload

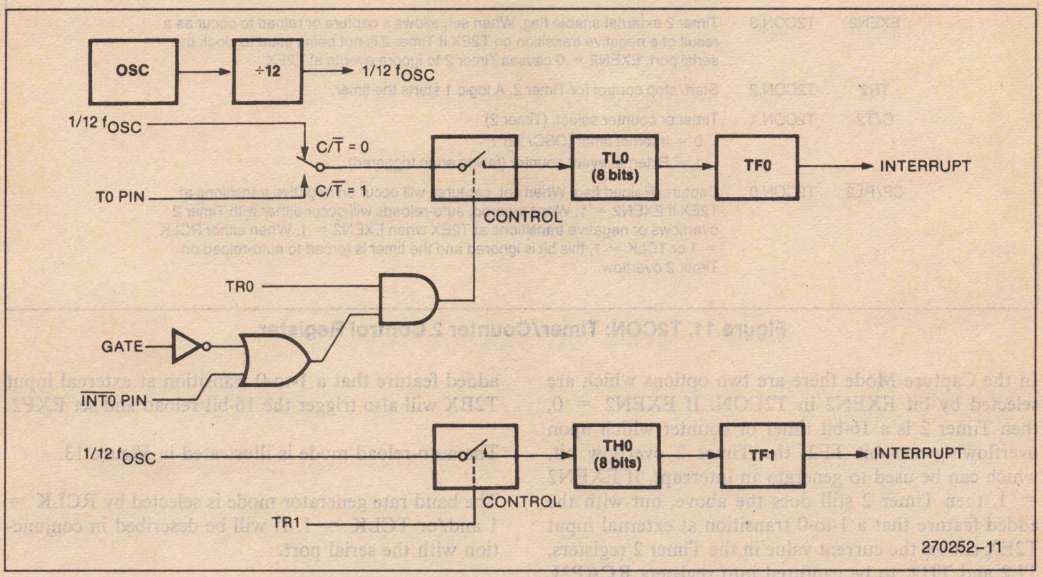


Figure 10. Timer/Counter 0 Mode 3: Two 8-Bit Counters

Timer 2

Timer 2 is a 16-bit Timer/Counter which is present only in the 8052. Like Timers 0 and 1, it can operate either as a timer or as an event counter. This is selected by bit C/$\overline{T2}$ in the Special Function Register T2CON (Figure 11). It has three operating modes: "capture," "auto-load" and "baud rate generator," which are selected by bits in T2CON as shown in Table 2.

Table 2. Timer 2 Operating Modes

RCLK + TCLK	CP/$\overline{RL2}$	TR2	Mode
0	0	1	16-bit Auto-Reload
0	1	1	16-bit Capture
1	X	1	Baud Rate Generator
X	X	0	(off)

(MSB)							(LSB)
TF2	EXF2	RCLK	TCLK	EXEN2	TR2	C/$\overline{T2}$	CP/$\overline{RL2}$

Symbol	Position	Name and Significance
TF2	T2CON.7	Timer 2 overflow flag set by a Timer 2 overflow and must be cleared by software. TF2 will not be set when either RCLK = 1 or TCLK = 1.
EXF2	T2CON.6	Timer 2 external flag set when either a capture or reload is caused by a negative transition on T2EX and EXEN2 = 1. When Timer 2 interrupt is enabled, EXF2 = 1 will cause the CPU to vector to the Timer 2 interrupt routine. EXF2 must be cleared by software.
RCLK	T2CON.5	Receive clock flag. When set, causes the serial port to use Timer 2 overflow pulses for its receive clock in Modes 1 and 3. RCLK = 0 causes Timer 1 overflow to be used for the receive clock.
TCLK	T2CON.4	Transmit clock flag. When set, causes the serial port to use Timer 2 overflow pulses for its transmit clock in modes 1 and 3. TCLK = 0 causes Timer 1 overflows to be used for the transmit clock.
EXEN2	T2CON.3	Timer 2 external enable flag. When set, allows a capture or reload to occur as a result of a negative transition on T2EX if Timer 2 is not being used to clock the serial port. EXEN2 = 0 causes Timer 2 to ignore events at T2EX.
TR2	T2CON.2	Start/stop control for Timer 2. A logic 1 starts the timer.
C/$\overline{T2}$	T2CON.1	Timer or counter select. (Timer 2) 0 = Internal timer (OSC/12) 1 = External event counter (falling edge triggered).
CP/$\overline{RL2}$	T2CON.0	Capture/Reload flag. When set, captures will occur on negative transitions at T2EX if EXEN2 = 1. When cleared, auto-reloads will occur either with Timer 2 overflows or negative transitions at T2EX when EXEN2 = 1. When either RCLK = 1 or TCLK = 1, this bit is ignored and the timer is forced to auto-reload on Timer 2 overflow.

Figure 11. T2CON: Timer/Counter 2 Control Register

In the Capture Mode there are two options which are selected by bit EXEN2 in T2CON. If EXEN2 = 0, then Timer 2 is a 16-bit timer or counter which upon overflowing sets bit TF2, the Timer 2 overflow bit, which can be used to generate an interrupt. If EXEN2 = 1, then Timer 2 still does the above, but with the added feature that a 1-to-0 transition at external input T2EX causes the current value in the Timer 2 registers, TL2 and TH2, to be captured into registers RCAP2L and RCAP2H, respectively. (RCAP2L and RCAP2H are new Special Function Registers in the 8052.) In addition, the transition at T2EX causes bit EXF2 in T2CON to be set, and EXF2, like TF2, can generate an interrupt.

The Capture Mode is illustrated in Figure 12.

In the auto-reload mode there are again two options, which are selected by bit EXEN2 in T2CON. If EXEN2 = 0, then when Timer 2 rolls over it not only sets TF2 but also causes the Timer 2 registers to be reloaded with the 16-bit value in registers RCAP2L and RCAP2H, which are preset by software. If EXEN2 = 1, then Timer 2 still does the above, but with the added feature that a 1-to-0 transition at external input T2EX will also trigger the 16-bit reload and set EXF2.

The auto-reload mode is illustrated in Figure 13.

The baud rate generator mode is selected by RCLK = 1 and/or TCLK = 1. It will be described in conjunction with the serial port.

SERIAL INTERFACE

The serial port is full duplex, meaning it can transmit and receive simultaneously. It is also receive-buffered, meaning it can commence reception of a second byte before a previously received byte has been read from the receive register. (However, if the first byte still hasn't been read by the time reception of the second byte is complete, one of the bytes will be lost). The serial port receive and transmit registers are both accessed at Special Function Register SBUF. Writing to SBUF loads the transmit register, and reading SBUF accesses a physically separate receive register.

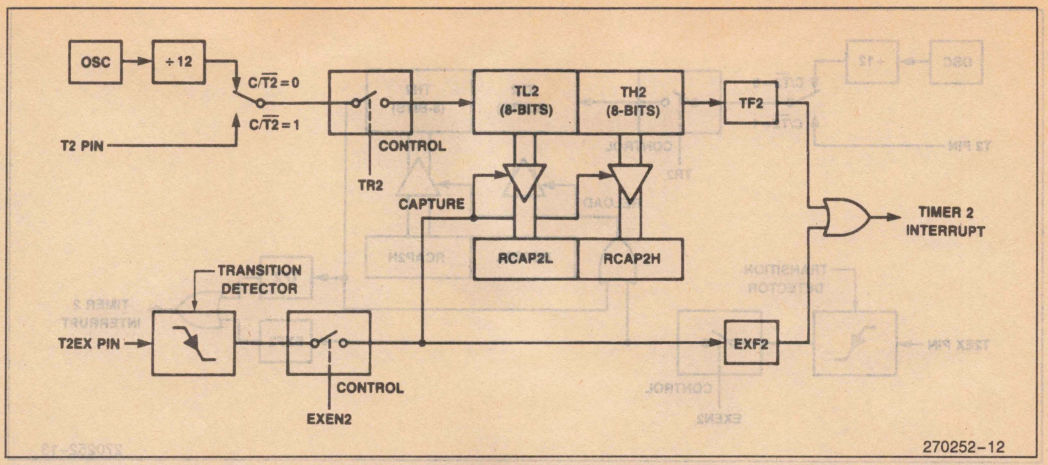


Figure 12. Timer 2 in Capture Mode

The serial port can operate in 4 modes:

Mode 0: Serial data enters and exits through RXD. TXD outputs the shift clock. 8 bits are transmitted/received: 8 data bits (LSB first). The baud rate is fixed at 1/12 the oscillator frequency.

Mode 1: 10 bits are transmitted (through TXD) or received (through RXD): a start bit (0), 8 data bits (LSB first), and a stop bit (1). On receive, the stop bit goes into RB8 in Special Function Register SCON. The baud rate is variable.

Mode 2: 11 bits are transmitted (through TXD) or received (through RXD): a start bit (0), 8 data bits (LSB first), a programmable 9th data bit, and a stop bit (1). On Transmit, the 9th data bit (TB8 in SCON) can be assigned the value of 0 or 1. Or, for example, the parity bit (P, in the PSW) could be moved into TB8. On receive, the 9th data bit goes into RB8 in Special Functon Register SCON, while the stop bit is ignored. The baud rate is programmable to either $1/32$ or $1/64$ the oscillator frequency.

Mode 3: 11 bits are transmitted (through TXD) or received (through RXD): a start bit (0), 8 data bits (LSB first), a programmable 9th data bit and a stop bit (1). In fact, Mode 3 is the same as Mode 2 in all respects except the baud rate. The baud rate in Mode 3 is variable.

In all four modes, transmission is initiated by any instruction that uses SBUF as a destination register. Reception is initiated in Mode 0 by the condition RI = 0 and REN = 1. Reception is initiated in the other modes by the incoming start bit if REN = 1.

Multiprocessor Communications

Modes 2 and 3 have a special provision for multiprocessor communications. In these modes, 9 data bits are received. The 9th one goes into RB8. Then comes a stop bit. The port can be programmed such that when the stop bit is received, the serial port interrupt will be activated only if RB8 = 1. This feature is enabled by setting bit SM2 in SCON. A way to use this feature in multiprocessor systems is as follows.

When the master processor wants to transmit a block of data to one of several slaves, it first sends out an address byte which identifies the target slave. An address byte differs from a data byte in that the 9th bit is 1 in an address byte and 0 in a data byte. With SM2 = 1, no slave will be interrupted by a data byte. An address byte, however, will interrupt all slaves, so that each slave can examine the received byte and see if it is being addressed. The addressed slave will clear its SM2 bit and prepare to receive the data bytes that will be coming. The slaves that weren't being addressed leave their SM2s set and go on about their business, ignoring the coming data bytes.

SM2 has no effect in Mode 0, and in Mode 1 can be used to check the validity of the stop bit. In a Mode 1 reception, if SM2 = 1, the receive interrupt will not be activated unless a valid stop bit is received.

Serial Port Control Register

The serial port control and status register is the Special Function Register SCON, shown in Figure 14. This register contains not only the mode selection bits, but also the 9th data bit for transmit and receive (TB8 and RB8), and the serial port interrupt bits (TI and RI).

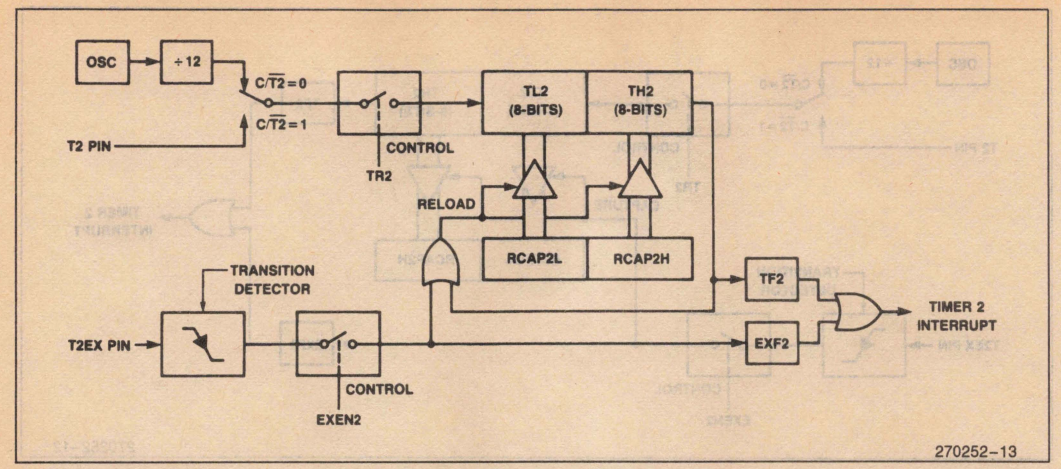


Figure 13. Timer 2 in Auto-Reload Mode

(MSB)							(LSB)
SM0	SM1	SM2	REN	TB8	RB8	TI	RI

Where SM0, SM1 specify the serial port mode, as follows:

SM0	SM1	Mode	Description	Baud Rate
0	0	0	shift register	$f_{osc.}/12$
0	1	1	8-bit UART	variable
1	0	2	9-bit UART	$f_{osc.}/64$ or $f_{osc.}/32$
1	1	3	9-bit UART	variable

- SM2 enables the multiprocessor communication feature in Modes 2 and 3. In Mode 2 or 3, if SM2 is set to 1 then RI will not be activated if the received 9th data bit (RB8) is 0. In Mode 1, if SM2 = 1 then RI will not be activated if a valid stop bit was not received. In Mode 0, SM2 should be 0.
- REN enables serial reception. Set by software to enable reception. Clear by software to disable reception.
- TB8 is the 9th data bit that will be transmitted in Modes 2 and 3. Set or clear by software as desired.
- RB8 in Modes 2 and 3, is the 9th data bit that was received. In Mode 1, if SM2 = 0, RB8 is the stop bit that was received. In Mode 0, RB8 is not used.
- TI is transmit interrupt flag. Set by hardware at the end of the 8th bit time in Mode 0, or at the beginning of the stop bit in the other modes, in any serial transmission. Must be cleared by software.
- RI is receive interrupt flag. Set by hardware at the end of the 8th bit time in Mode 0, or halfway through the stop bit time in the other modes, in any serial reception (except see SM2). Must be cleared by software.

Figure 14. SCON: Serial Port Control Register

Baud Rates

The baud rate in Mode 0 is fixed:

$$\text{Mode 0 Baud Rate} = \frac{\text{Oscillator Frequency}}{12}$$

The baud rate in Mode 2 depends on the value of bit SMOD in Special Function Register PCON. If SMOD = 0 (which is the value on reset), the baud rate $1/64$ the oscillator frequency. If SMOD = 1, the baud rate is $1/32$ the oscillator frequency.

$$\text{Mode 2 Baud Rate} = \frac{2^{\text{SMOD}}}{64} \times (\text{Oscillator Frequency})$$

In the 8051, the baud rates in Modes 1 and 3 are determined by the Timer 1 overflow rate. In the 8052, these baud rates can be determined by Timer 1, or by Timer 2, or by both (one for transmit and the other for receive).

Using Timer 1 to Generate Baud Rates

When Timer 1 is used as the baud rate generator, the baud rates in Modes 1 and 3 are determined by the Timer 1 overflow rate and the value of SMOD as follows:

$$\text{Modes 1, 3 Baud Rate} = \frac{2^{SMOD}}{32} \times \text{(Timer 1 Overflow Rate)}$$

The Timer 1 interrupt should be disabled in this application. The Timer itself can be configured for either "timer" or "counter" operation, and in any of its 3 running modes. In the most typical applications, it is configured for "timer" operation, in the auto-reload mode (high nibble of TMOD = 0010B). In that case, the baud rate is given by the formula

$$\text{Modes 1, 3 Baud Rate} = \frac{2^{SMOD}}{32} \times \frac{\text{Oscillator Frequency}}{12x\,[256 - (TH1)]}$$

One can achieve very low baud rates with Timer 1 by leaving the Timer 1 interrupt enabled, and configuring the Timer to run as a 16-bit timer (high nibble of TMOD = 0001B), and using the Timer 1 interrupt to do a 16-bit software reload.

Figure 15 lists various commonly used baud rates and how they can be obtained from Timer 1.

Baud Rate	f_{osc}	SMOD	Timer 1		
			C/$\overline{T}$	Mode	Reload Value
Mode 0 Max: 1 MHZ	12 MHZ	X	X	X	X
Mode 2 Max: 375K	12 MHZ	1	X	X	X
Modes 1, 3: 62.5K	12 MHZ	1	0	2	FFH
19.2K	11.059 MHZ	1	0	2	FDH
9.6K	11.059 MHZ	0	0	2	FDH
4.8K	11.059 MHZ	0	0	2	FAH
2.4K	11.059 MHZ	0	0	2	F4H
1.2K	11.059 MHZ	0	0	2	E8H
137.5K	11.986 MHZ	0	0	2	1DH
110K	6 MHZ	0	0	2	72H
110K	12 MHZ	0	0	1	FEEBH

Figure 15. Timer 1 Generated Commonly Used Baud Rates

Using Timer 2 to Generate Baud Rates

In the 8052, Timer 2 is selected as the baud rate generator by setting TCLK and/or RCLK in T2CON (Figure 11). Note then the baud rates for transmit and receive can be simultaneously different. Setting RCLK and/or TCLK puts Timer 2 into its baud rate generator mode, as shown in Figure 16.

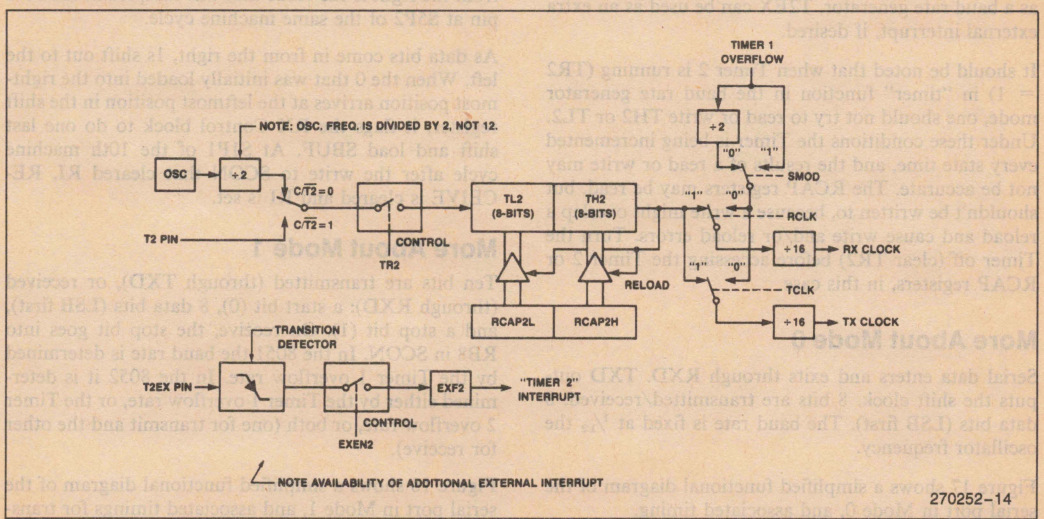


Figure 16. Timer 2 in Baud Rate Generator Mode

The baud rate generator mode is similar to the auto-reload mode, in that a rollover in TH2 causes the Timer 2 registers to be reloaded with the 16-bit value in registers RCAP2H and RCAP2L, which are preset by software.

Now, the baud rates in Modes 1 and 3 are determined by Timer 2's overflow rate as follows:

$$\text{Modes 1, 3 Baud Rate} = \frac{\text{Timer 2 Overflow Rate}}{16}$$

The Timer can be configured for either "timer" or "counter" operation. In the most typical applications, it is configured for "timer" operation (C/T2 = 0). "Timer" operation is a little different for Timer 2 when it's being used as a baud rate generator. Normally, as a timer it would increment every machine cycle (thus at $1/12$ the oscillator frequency). As a baud rate generator, however, it increments every state time (thus at $1/2$ the oscillator frequency). In that case the baud rate is given by the formula

$$\text{Modes 1, 3 Baud Rate} = \frac{\text{Oscillator Frequency}}{32\text{x}\,[65536 - (\text{RCAP2H, RCAP2L})]}$$

where (RCAP2H, RCAP2L) is the content of RCAP2H and RCAP2L taken as a 16-bit unsigned integer.

Timer 2 as a baud rate generator is shown in Figure 16. This Figure is valid only if RCLK + TCLK = 1 in T2CON. Note that a rollover in TH2 does not set TF2, and will not generate an interrupt. Therefore, the Timer 2 interrupt does not have to be disabled when Timer 2 is in the baud rate generator mode. Note too, that if EXEN2 is set, a 1-to-0 transition in T2EX will set EXF2 but will not cause a reload from (RCAP2H, RCAP2L) to (TH2, TL2). Thus when Timer 2 is in use as a baud rate generator, T2EX can be used as an extra external interrupt, if desired.

It should be noted that when Timer 2 is running (TR2 = 1) in "timer" function in the baud rate generator mode, one should not try to read or write TH2 or TL2. Under these conditions the Timer is being incremented every state time, and the results of a read or write may not be accurate. The RCAP registers may be read, but shouldn't be written to, because a write might overlap a reload and cause write and/or reload errors. Turn the Timer off (clear TR2) before accessing the Timer 2 or RCAP registers, in this case.

More About Mode 0

Serial data enters and exits through RXD. TXD outputs the shift clock. 8 bits are transmitted/received: 8 data bits (LSB first). The baud rate is fixed at $1/12$ the oscillator frequency.

Figure 17 shows a simplified functional diagram of the serial port in Mode 0, and associated timing.

Transmission is initiated by any instruction that uses SBUF as a destination register. The "write to SBUF" signal at S6P2 also loads a 1 into the 9th position of the transmit shift register and tells the TX Control block to commence a transmission. The internal timing is such that one full machine cycle will elapse between "write to SBUF," and activation of SEND.

SEND enables the output of the shift register to the alternate output function line of P3.0, and also enables SHIFT CLOCK to the alternate output function line of P3.1. SHIFT CLOCK is low during S3, S4, and S5 of every machine cycle, and high during S6, S1 and S2. At S6P2 of every machine cycle in which SEND is active, the contents of the transmit shift register are shifted to the right one position.

As data bits shift out to the right, zeroes come in from the left. When the MSB of the data byte is at the output position of the shift register, then the 1 that was initially loaded into the 9th position, is just to the left of the MSB, and all positions to the left of that contain zeroes. This condition flags the TX Control block to do one last shift and then deactivate SEND and set TI. Both of these actions occur at S1P1 of the 10th machine cycle after "write to SBUF."

Reception is initiated by the condition REN = 1 and R1 = 0. At S6P2 of the next machine cycle, the RX Control unit writes the bits 11111110 to the receive shift register, and in the next clock phase activates RECEIVE.

RECEIVE enables SHIFT CLOCK to the alternate output function line of P3.1. SHIFT CLOCK makes transitions at S3P1 and S6P1 of every machine cycle. At S6P2 of every machine cycle in which RECEIVE is active, the contents of the receive shift register are shifted to the left one position. The value that comes in from the right is the value that was sampled at the P3.0 pin at S5P2 of the same machine cycle.

As data bits come in from the right, 1s shift out to the left. When the 0 that was initially loaded into the rightmost position arrives at the leftmost position in the shift register, it flags the RX Control block to do one last shift and load SBUF. At S1P1 of the 10th machine cycle after the write to SCON that cleared RI, RECEIVE is cleared and RI is set.

More About Mode 1

Ten bits are transmitted (through TXD), or received (through RXD): a start bit (0), 8 data bits (LSB first), and a stop bit (1). On receive, the stop bit goes into RB8 in SCON. In the 8051 the baud rate is determined by the Timer 1 overflow rate. In the 8052 it is determined either by the Timer 1 overflow rate, or the Timer 2 overflow rate, or both (one for transmit and the other for receive).

Figure 18 shows a simplified functional diagram of the serial port in Mode 1, and associated timings for transmit receive.

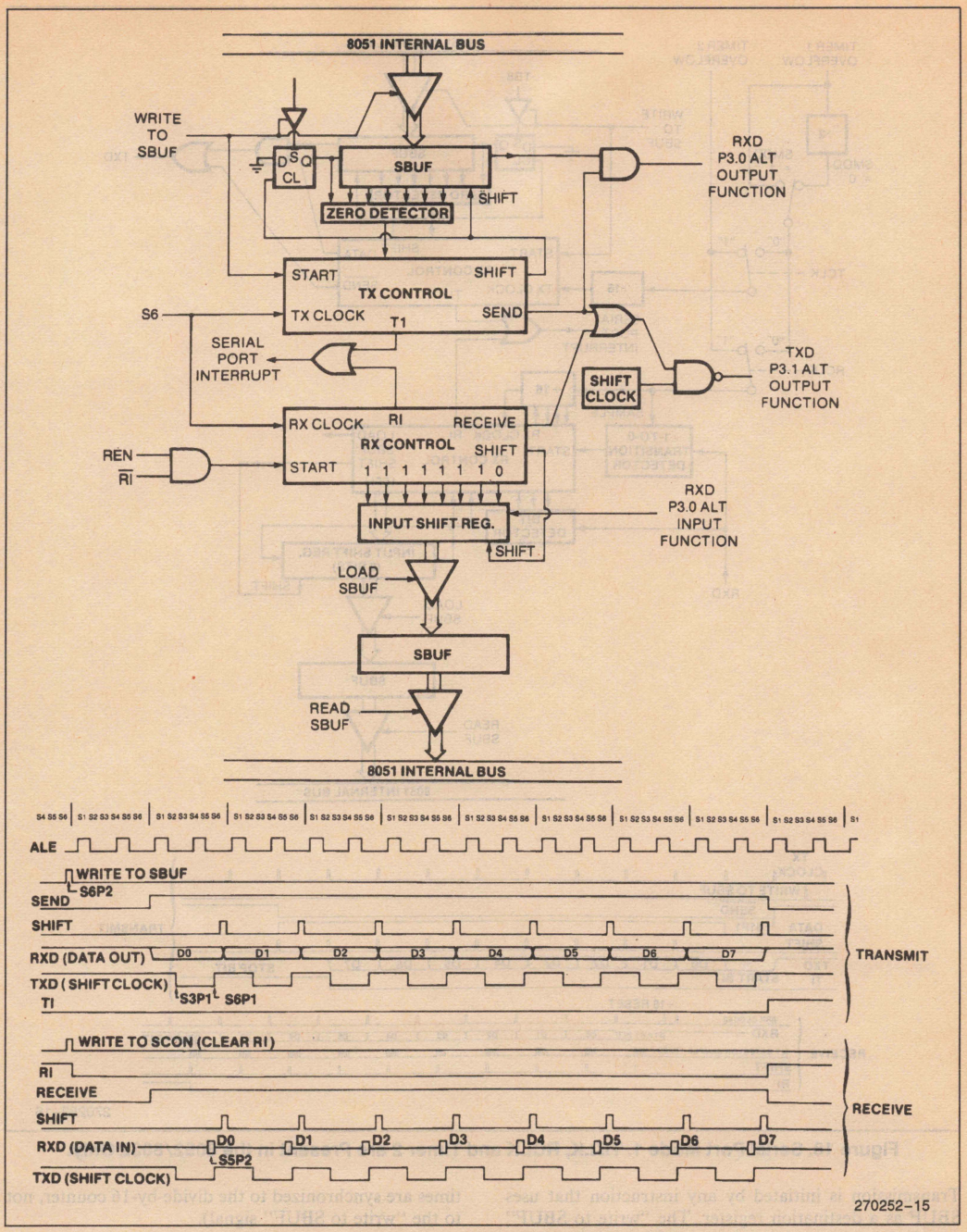


Figure 17. Serial Port Mode 0

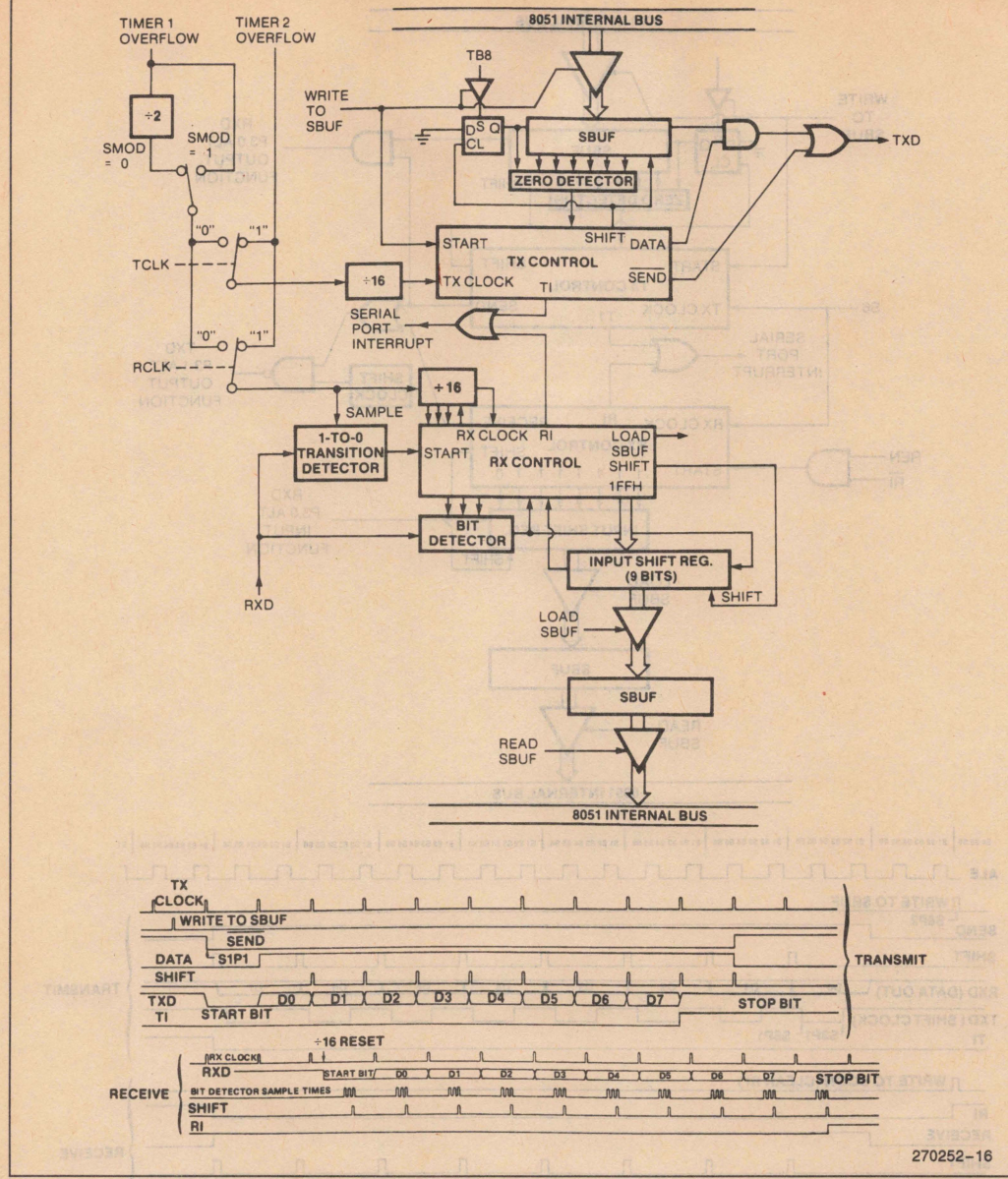


Figure 18. Serial Port Mode 1. TCLK, RCLK and Timer 2 are Present in the 8052/8032 Only.

Transmission is initiated by any instruction that uses SBUF as a destination register. The "write to SBUF" signal also loads a 1 into the 9th bit position of the transmit shift register and flags the TX Control unit that a transmission is requested. Transmission actually commences at S1P1 of the machine cycle following the next rollover in the divide-by-16 counter. (Thus, the bit times are synchronized to the divide-by-16 counter, not to the "write to SBUF" signal).

The transmission begins with activation of $\overline{\text{SEND}}$, which puts the start bit at TXD. One bit time later, DATA is activated, which enables the output bit of the transmit shift register to TXD. The first shift pulse occurs one bit time after that.

As data bits shift out to the right, zeroes are clocked in from the left. When the MSB of the data byte is at the output position of the shift register, then the 1 that was initially loaded into the 9th position is just to the left of the MSB, and all positions to the left of that contain zeroes. This condition flags the TX Control unit to do one last shift and then deactivate SEND and set TI. This occurs at the 10th divide-by-16 rollover after "write to SBUF."

Reception is initiated by a detected 1-to-0 transition at RXD. For this purpose RXD is sampled at a rate of 16 times whatever baud rate has been established. When a transition is detected, the divide-by-16 counter is immediately reset, and 1FFH is written into the input shift register. Resetting the divide-by-16 counter aligns its rollovers with the boundaries of the incoming bit times.

The 16 states of the counter divide each bit time into 16ths. At the 7th, 8th, and 9th counter states of each bit time, the bit detector samples the value of RXD. The value accepted is the value that was seen in at least 2 of the 3 samples. This is done for noise rejection. If the value accepted during the first bit time is not 0, the receive circuits are reset and the unit goes back to looking for another 1-to-0 transition. This is to provide rejection of false start bits. If the start bit proves valid, it is shifted into the input shift register, and reception of the rest of the frame will proceed.

As data bits come in from the right, 1s shift out to the left. When the start bit arrives at the leftmost position in the shift register, (which in mode 1 is a 9-bit register), it flags the RX Control block to do one last shift, load SBUF and RB8, and set RI. The signal to load SBUF and RB8, and to set RI, will be generated if, and only if, the following conditions are met at the time the final shift pulse is generated.

1) RI = 0, and
2) Either SM2 = 0, or the received stop bit = 1

If either of these two conditions is not met, the received frame is irretrievably lost. If both conditions are met, the stop bit goes into RB8, the 8 data bits go into SBUF, and RI is activated. At this time, whether the above conditions are met or not, the unit goes back to looking for a 1-to-0 transition in RXD.

More About Modes 2 and 3

Eleven bits are transmitted (through TXD), or received (through RXD): a start bit (0), 8 data bits (LSB first), a programmable 9th data bit, and a stop bit (1). On transmit, the 9th data bit (TB8) can be assigned the value of 0 or 1. On receive, the 9th data bit goes into RB8 in SCON. The baud rate is programmable to either $1/32$ or $1/64$ the oscillator frequency in Mode 2. Mode 3 may have a variable baud rate generated from either Timer 1 or 2 depending on the state of TCLK and RCLK.

Figures 19 and 20 show a functional diagram of the serial port in Modes 2 and 3. The receive portion is exactly the same as in Mode 1. The transmit portion differs from Mode 1 only in the 9th bit of the transmit shift register.

Transmission is initiated by any instruction that uses SBUF as a destination register. The "write to SBUF" signal also loads TB8 into the 9th bit position of the transmit shift register and flags the TX Control unit that a transmission is requested. Transmission commences at S1P1 of the machine cycle following the next rollover in the divide-by-16 counter. (Thus, the bit times are synchronized to the divide-by-16 counter, not to the "write to SBUF" signal.)

The transmission begins with activation of SEND, which puts the start bit at TXD. One bit time later, DATA is activated, which enables the output bit of the transmit shift register to TXD. The first shift pulse occurs one bit time after that. The first shift clocks a 1 (the stop bit) into the 9th bit position of the shift register. Thereafter, only zeroes are clocked in. Thus, as data bits shift out to the right, zeroes are clocked in from the left. When TB8 is at the output position of the shift register, then the stop bit is just to the left of TB8, and all positions to the left of that contain zeroes. This condition flags the TX Control unit to do one last shift and then deactivate SEND and set TI. This occurs at the 11th divide-by-16 rollover after "write to SBUF."

Reception is initiated by a detected 1-to-0 transition at RXD. For this purpose RXD is sampled at a rate of 16 times whatever baud rate has been established. When a transition is detected, the divide-by-16 counter is immediately reset, and 1FFH is written to the input shift register.

At the 7th, 8th and 9th counter states of each bit time, the bit detector samples the value of RXD. The value accepted is the value that was seen in at least 2 of the 3 samples. If the value accepted during the first bit time is not 0, the receive circuits are reset and the unit goes back to looking for another 1-to-0 transition. If the start bit proves valid, it is shifted into the input shift register, and reception of the rest of the frame will proceed.

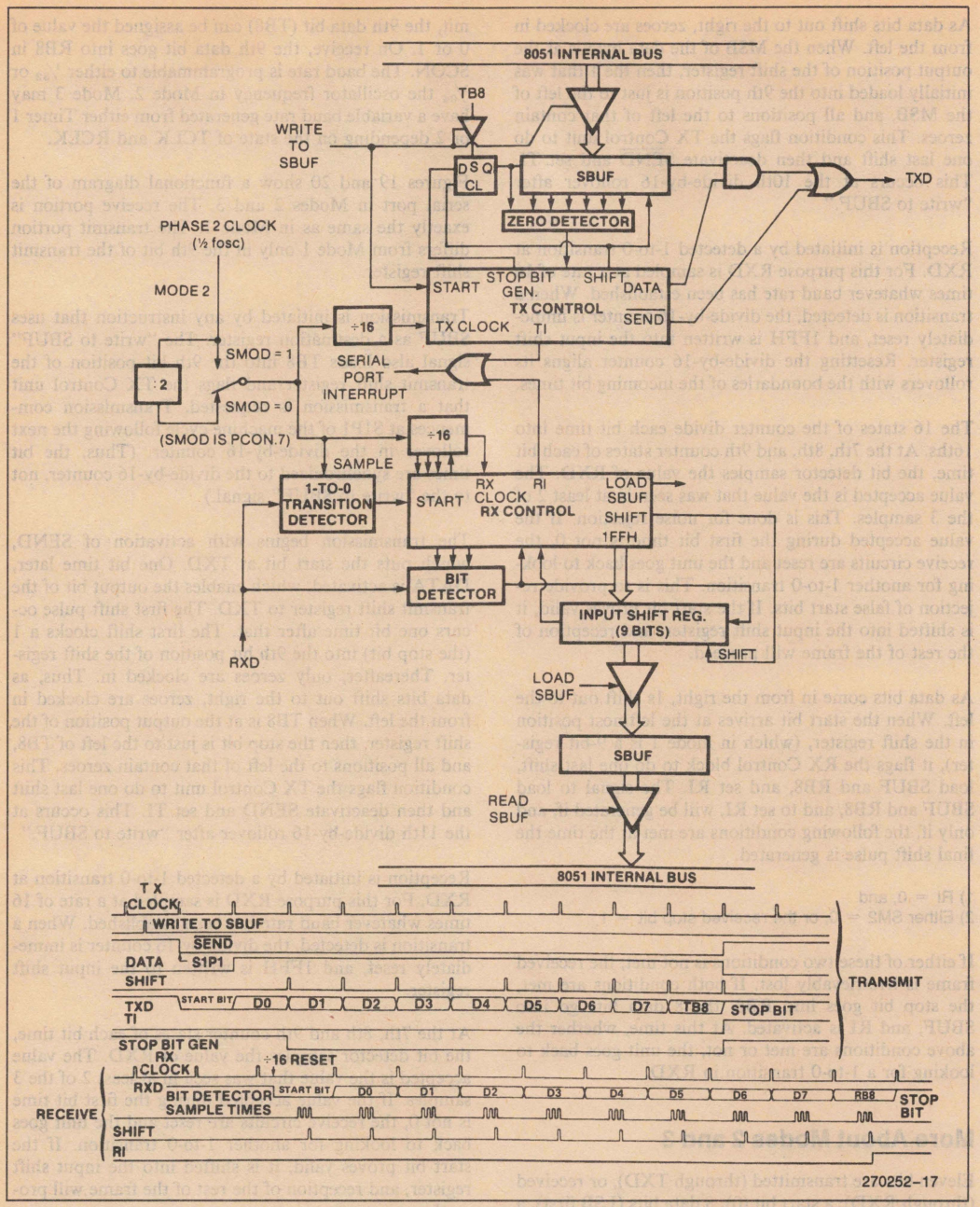


Figure 19. Serial Port Mode 2

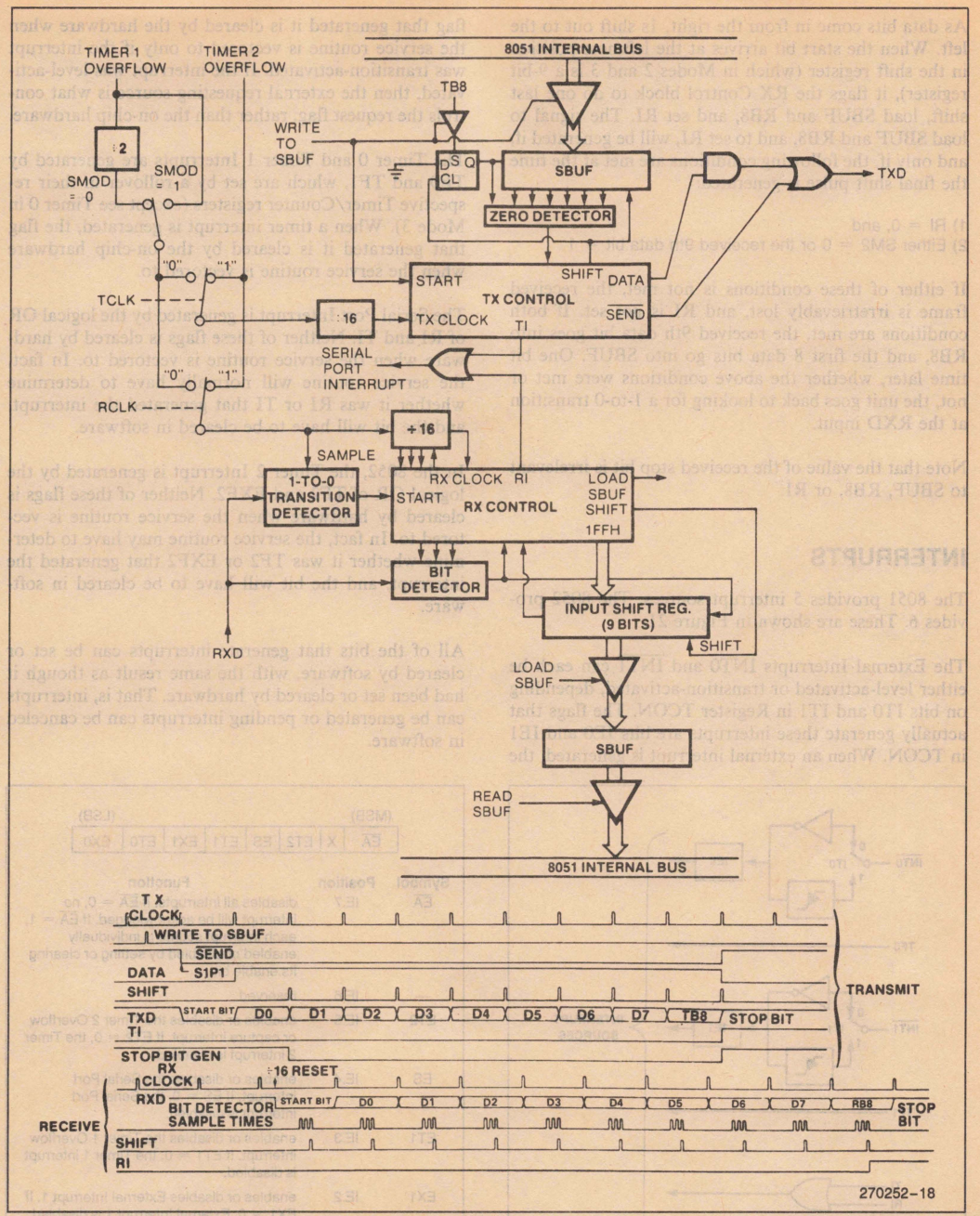


Figure 20. Serial Port Mode 3. TCLK, RCLK, and Timer 2 are Present in the 8052/8032 Only.

As data bits come in from the right, 1s shift out to the left. When the start bit arrives at the leftmost position in the shift register (which in Modes 2 and 3 is a 9-bit register), it flags the RX Control block to do one last shift, load SBUF and RB8, and set RI. The signal to load SBUF and RB8, and to set RI, will be generated if, and only if, the following conditions are met at the time the final shift pulse is generated:

1) RI = 0, and
2) Either SM2 = 0 or the received 9th data bit = 1

If either of these conditions is not met, the received frame is irretrievably lost, and RI is not set. If both conditions are met, the received 9th data bit goes into RB8, and the first 8 data bits go into SBUF. One bit time later, whether the above conditions were met or not, the unit goes back to looking for a 1-to-0 transition at the RXD input.

Note that the value of the received stop bit is irrelevant to SBUF, RB8, or RI.

INTERRUPTS

The 8051 provides 5 interrupt sources. The 8052 provides 6. These are shown in Figure 21.

The External Interrupts $\overline{INT0}$ and $\overline{INT1}$ can each be either level-activated or transition-activated, depending on bits IT0 and IT1 in Register TCON. The flags that actually generate these interrupts are bits IE0 and IE1 in TCON. When an external interrupt is generated, the flag that generated it is cleared by the hardware when the service routine is vectored to only if the interrupt was transition-activated. If the interrupt was level-activated, then the external requesting source is what controls the request flag, rather than the on-chip hardware.

The Timer 0 and Timer 1 Interrupts are generated by TF0 and TF1, which are set by a rollover in their respective Timer/Counter registers (except see Timer 0 in Mode 3). When a timer interrupt is generated, the flag that generated it is cleared by the on-chip hardware when the service routine is vectored to.

The Serial Port Interrupt is generated by the logical OR of RI and TI. Neither of these flags is cleared by hardware when the service routine is vectored to. In fact, the service routine will normally have to determine whether it was RI or TI that generated the interrupt, and the bit will have to be cleared in software.

In the 8052, the Timer 2 Interrupt is generated by the logical OR of TF2 and EXF2. Neither of these flags is cleared by hardware when the service routine is vectored to. In fact, the service routine may have to determine whether it was TF2 or EXF2 that generated the interrupt, and the bit will have to be cleared in software.

All of the bits that generate interrupts can be set or cleared by software, with the same result as though it had been set or cleared by hardware. That is, interrupts can be generated or pending interrupts can be canceled in software.

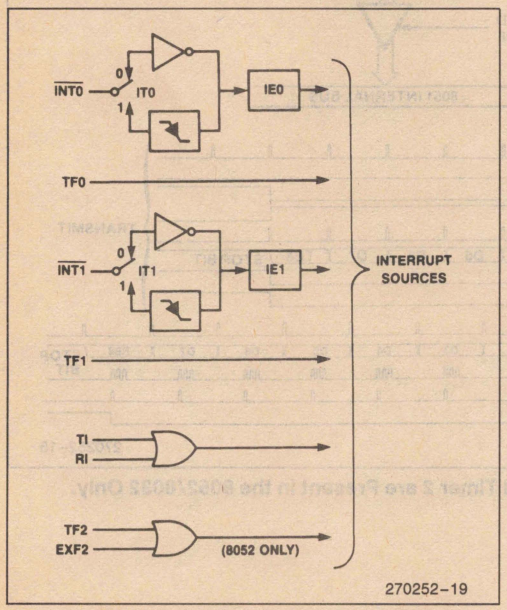


Figure 21. MCS®-51 Interrupt Sources

(MSB)							(LSB)
$\overline{EA}$	X	ET2	ES	ET1	EX1	ET0	EX0

Symbol	Position	Function
$\overline{EA}$	IE.7	disables all interrupts. If $\overline{EA}$ = 0, no interrupt will be acknowledged. If $\overline{EA}$ = 1, each interrupt source is individually enabled or disabled by setting or clearing its enable bit.
—	IE.6	reserved.
ET2	IE.5	enables or disables the Timer 2 Overflow or capture interrupt. If ET2 = 0, the Timer 2 interrupt is disabled.
ES	IE.4	enables or disables the Serial Port interrupt. If ES = 0, the Serial Port interrupt is disabled.
ET1	IE.3	enables or disables the Timer 1 Overflow interrupt. If ET1 = 0, the Timer 1 interrupt is disabled.
EX1	IE.2	enables or disables External Interrupt 1. If EX1 = 0, External Interrupt 1 is disabled.
ET0	IE.1	enables or disables the Timer 0 Overflow interrupt. If ET0 = 0, the Timer 0 interrupt is disabled.
EX0	IE.0	enables or disables External Interrupt 0. If EX0 = 0, External Interrupt 0 is disabled.

Figure 22. IE: Interrupt Enable Register

Each of these interrupt sources can be individually enabled or disabled by setting or clearing a bit in Special Function Register IE (Figure 22). IE contains also a global disable bit, $\overline{EA}$, which disables all interrupts at once.

Note in Figure 23 that bit position IE.6 is unimplemented. In the 8051s, bit position IE.5 is also unimplemented. User software should not write 1s to these bit positions, since they may be used in future MCS-51 products.

Priority Level Structure

Each interrupt source can also be individually programmed to one of two priority levels by setting or clearing a bit in Special Function Register IP (Figure 23). A low-priority interrupt can itself be interrupted by a high-priority interrupt, but not by another low-priority interrupt. A high-priority interrupt can't be interrupted by any other interrupt source.

(MSB) (LSB)

X	X	PT2	PS	PT1	PX1	PT0	PX0

Symbol	Position	Function
—	IP.7	reserved
—	IP.6	reserved
PT2	IP.5	defines the Timer 2 interrupt priority level. PT2 = 1 programs it to the higher priority level.
PS	IP.4	defines the Serial Port interrupt priority level. PS = 1 programs it to the higher priority level.
PT1	IP.3	defines the Timer 1 interrupt priority level. PT1 = 1 programs it to the higher priority level.
PT0	IP.1	defines the Timer 0 interrupt priority level. PT0 = 1 programs it to the higher priority level.
PX0	IP.0	defines the External Interrupt 0 priority level. PX0 = 1 programs it to the higher priority level.

Figure 23. IP: Interrupt Priority Register

If two requests of different priority levels are received simultaneously, the request of higher priority level is serviced. If requests of the **same** priority level are received simultaneously, an internal polling sequence determines which request is serviced. Thus within each priority level there is a second priority structure determined by the polling sequence, as follows:

	Source	Priority Within Level
1.	IE0	(highest)
2.	TF0	
3.	IE1	
4.	TF1	
5.	RI + TI	
6.	TF2 + EXF2	(lowest)

Note that the "priority within level" structure is only used to resolve *simultaneous requests of the same priority level.*

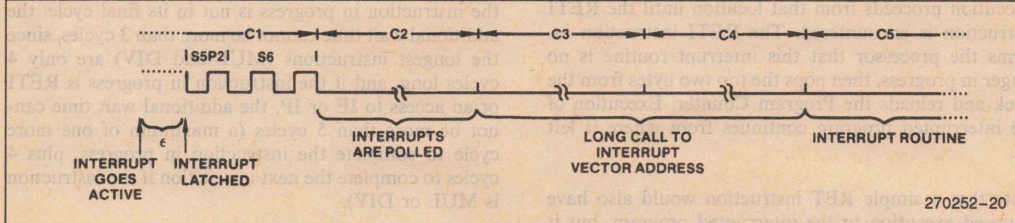
The IP register contains a number of unimplemented bits. IP.7 and IP.6 are vacant in the 8052s, and in the 8051s these and IP.5 are vacant. User software should not write 1s to these bit positions, since they may be used in future MCS-51 products.

How Interrupts Are Handled

The interrupt flags are sampled at S5P2 of every machine cycle. The samples are polled during the following machine cycle. If one of the flags was in a set condition at S5P2 of the preceding cycle, the polling cycle will find it and the interrupt system will generate an LCALL to the appropriate service routine, provided this hardware-generated LCALL is not blocked by any of the following conditions:

1. An interrupt of equal or higher priority level is already in progress.
2. The current (polling) cycle is not the final cycle in the execution of the instruction in progress.
3. The instruction in progress is RETI or any write to the IE or IP registers.

Any of these three conditions will block the generation of the LCALL to the interrupt service routine. Condition 2 ensures that the instruction in progress will be



This is the fastest possible response when C2 is the final cycle of an instruction other than RETI or an access to IE or IP.

Figure 24. Interrupt Response Timing Diagram

completed before vectoring to any service routine. Condition 3 ensures that if the instruction in progress is RETI or any access to IE or IP, then at least *one more* instruction will be executed before any interrupt is vectored to.

The polling cycle is repeated with each machine cycle, and the values polled are the values that were present at S5P2 of the previous machine cycle. Note then that if an interrupt flag is active but not being responded to for one of the above conditions, if the flag is not *still* active when the blocking condition is removed, the denied interrupt will not be serviced. In other words, the fact that the interrupt flag was once active but not serviced is not remembered. Every polling cycle is new.

The polling cycle/LCALL sequence is illustrated in Figure 24.

Note that if an interrupt of higher priority level goes active prior to S5P2 of the machine cycle labeled C3 in Figure 24, then in accordance with the above rules it will be vectored to during C5 and C6, without any instruction of the lower priority routine having been executed.

Thus the processor acknowledges an interrupt request by executing a hardware-generated LCALL to the appropriate servicing routine. In some cases it also clears the flag that generated the interrupt, and in other cases it doesn't. It never clears the Serial Port or Timer 2 flags. This has to be done in the user's software. It clears an external interrupt flag (IE0 or IE1) only if it was transition-activated. The hardware-generated LCALL pushes the contents of the Program Counter onto the stack (but it does not save the PSW) and reloads the PC with an address that depends on the source of the interrupt being vectored to, as shown below.

Source	Vector Address
IE0	0003H
TF0	000BH
IE1	0013H
TF1	001BH
RI + TI	0023H
TF2 + EXF2	002BH

Execution proceeds from that location until the RETI instruction is encountered. The RETI instruction informs the processor that this interrupt routine is no longer in progress, then pops the top two bytes from the stack and reloads the Program Counter. Execution of the interrupted program continues from where it left off.

Note that a simple RET instruction would also have returned execution to the interrupted program, but it would have left the interrupt control system thinking an interrupt was still in progress.

External Interrupts

The external sources can be programmed to be level-activated or transition-activated by setting or clearing bit IT1 or IT0 in Register TCON. If ITx = 0, external interrupt x is triggered by a detected low at the $\overline{\text{INTx}}$ pin. If ITx = 1, external interrupt x is edge-triggered. In this mode if successive samples of the $\overline{\text{INTx}}$ pin show a high in one cycle and a low in the next cycle, interrupt request flag IEx in TCON is set. Flag bit IEx then requests the interrupt.

Since the external interrupt pins are sampled once each machine cycle, an input high or low should hold for at least 12 oscillator periods to ensure sampling. If the external interrupt is transition-activated, the external source has to hold the request pin high for at least one cycle, and then hold it low for at least one cycle to ensure that the transition is seen so that interrupt request flag IEx will be set. IEx will be automatically cleared by the CPU when the service routine is called.

If the external interrupt is level-activated, the external source has to hold the request active until the requested interrupt is actually generated. Then it has to deactivate the request before the interrupt service routine is completed, or else another interrupt will be generated.

Response Time

The $\overline{\text{INT0}}$ and $\overline{\text{INT1}}$ levels are inverted and latched into IE0 and IE1 at S5P2 of every machine cycle. The values are not actually polled by the circuitry until the next machine cycle. If a request is active and conditions are right for it to be acknowledged, a hardware subroutine call to the requested service routine will be the next instruction to be executed. The call itself takes two cycles. Thus, a minimum of three complete machine cycles elapse between activation of an external interrupt request and the beginning of execution of the first instruction of the service routine. Figure 24 shows interrupt response timings.

A longer response time would result if the request is blocked by one of the 3 previously listed conditions. If an interrupt of equal or higher priority level is already in progress, the additional wait time obviously depends on the nature of the other interrupt's service routine. If the instruction in progress is not in its final cycle, the additional wait time cannot be more than 3 cycles, since the longest instructions (MUL and DIV) are only 4 cycles long, and if the instruction in progress is RETI or an access to IE or IP, the additional wait time cannot be more than 5 cycles (a maximum of one more cycle to complete the instruction in progress, plus 4 cycles to complete the next instruction if the instruction is MUL or DIV).

Thus, in a single-interrupt system, the response time is always more than 3 cycles and less than 9 cycles.

SINGLE-STEP OPERATION

The 8051 interrupt structure allows single-step execution with very little software overhead. As previously noted, an interrupt request will not be responded to while an interrupt of equal priority level is still in progress, nor will it be responded to after RETI until at least one other instruction has been executed. Thus, once an interrupt routine has been entered, it cannot be re-entered until at least one instruction of the interrupted program is executed. One way to use this feature for single-stop operation is to program one of the external interrupts (say, $\overline{INT0}$) to be level-activated. The service routine for the interrupt will terminate with the following code:

```
JNB   P3.2,$  ;Wait Here Till INT0 Goes High
JB    P3.2,$  ;Now Wait Here Till it Goes Low
RETI          :Go Back and Execute One Instruction
```

Now if the $\overline{INT0}$ pin, which is also the P3.2 pin, is held normally low, the CPU will go right into the External Interrupt 0 routine and stay there until $\overline{INT0}$ is pulsed (from low to high to low). Then it will execute RETI, go back to the task program, execute one instruction, and immediately re-enter the External Interrupt 0 routine to await the next pulsing of P3.2. One step of the task program is executed each time P3.2 is pulsed.

RESET

The reset input is the RST pin, which is the input to a Schmitt Trigger.

A reset is accomplished by holding the RST pin high for at least two machine cycles (24 oscillator periods), *while the oscillator is running.* The CPU responds by generating an internal reset, with the timing shown in Figure 25.

The external reset signal is asynchronous to the internal clock. The RST pin is sampled during State 5 Phase 2 of every machine cycle. The port pins will maintain their current activities for 19 oscillator periods after a logic 1 has been sampled at the RST pin; that is, for 19 to 31 oscillator periods after the external reset signal has been applied to the RST pin.

The internal reset algorithm writes 0s to all the SFRs except the port latches, the Stack Pointer, and SBUF. The port latches are initialized to FFH, the Stack Pointer to 07H, and SBUF is indeterminate. Table 3 lists the SFRs and their reset values.

The internal RAM is not affected by reset. On power up the RAM content is indeterminate.

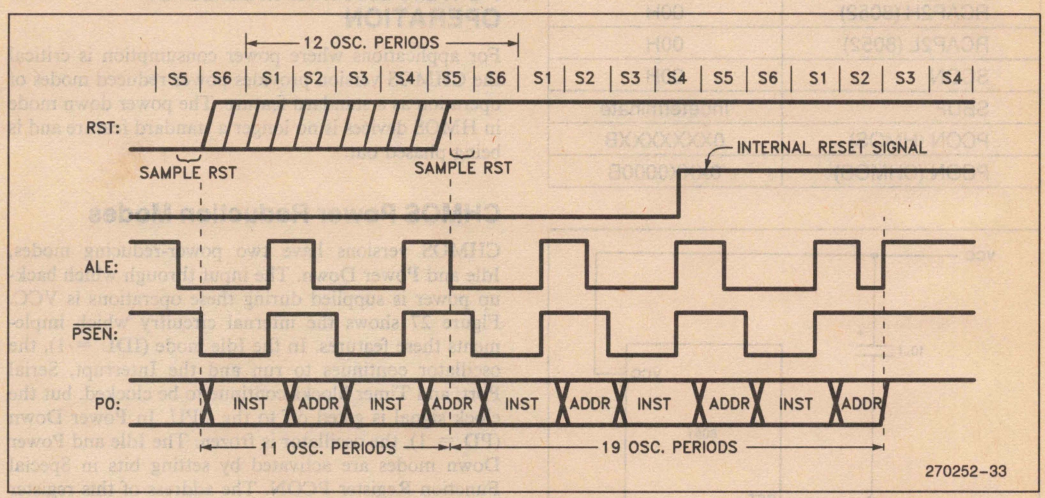


Figure 25. Reset Timing

Table 3. Reset Values of the SFRs

SFR Name	Reset Value
PC	0000H
ACC	00H
B	00H
PSW	00H
SP	07H
DPTR	0000H
P0–P3	FFH
IP (8051)	XXX00000B
IP (8052)	XX000000B
IE (8051)	0XX00000B
IE (8052)	0X000000B
TMOD	00H
TCON	00H
TH0	00H
TL0	00H
TH1	00H
TL1	00H
TH2 (8052)	00H
TL2 (8052)	00H
RCAP2H (8052)	00H
RCAP2L (8052)	00H
SCON	00H
SBUF	Indeterminate
PCON (HMOS)	0XXXXXXXB
PCON (CHMOS)	0XXX0000B

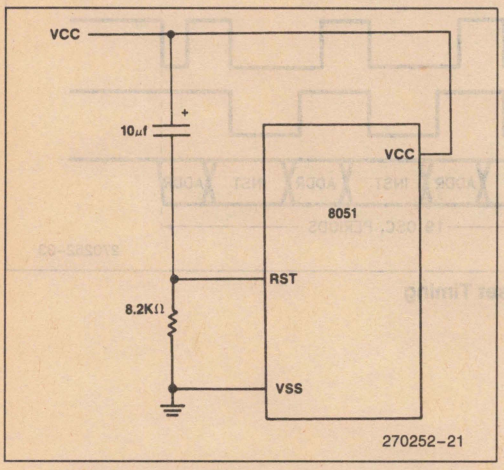


Figure 26. Power on Reset Circuit

POWER-ON RESET

An automatic reset can be obtained when VCC is turned on by connecting the RST pin to VCC through a 10 μf capacitor and to VSS through an 8.2 KΩ resistor, providing the VCC risetime does not exceed a millisecond and the oscillator start-up time does not exceed 10 milliseconds. This power-on reset circuit is shown in Figure 26. The CHMOS devices do not require the 8.2K pulldown resistor, although its presence does no harm.

When power is turned on the circuit holds the RST pin high for an amount of time that depends on the value of the capacitor and the rate at which it charges. To ensure a good reset the RST pin must be high long enough to allow the oscillator time to start up (normally a few msec) plus two machine cycles.

Note that the port pins will be in a random state until the oscillator has started and the internal reset algorithm has written 1s to them.

With this circuit, reducing VCC quickly to 0 causes the RST pin voltage to momentarily fall below 0V. However, this voltage is internally limited, and will not harm the device.

POWER-SAVING MODES OF OPERATION

For applications where power consumption is critical the CHMOS version provides power reduced modes of operation as a standard feature. The power down mode in HMOS devices is no longer a standard feature and is being phased out.

CHMOS Power Reduction Modes

CHMOS versions have two power-reducing modes, Idle and Power Down. The input through which back-up power is supplied during these operations is VCC. Figure 27 shows the internal circuitry which implements these features. In the Idle mode (IDL = 1), the oscillator continues to run and the Interrupt, Serial Port, and Timer blocks continue to be clocked, but the clock signal is gated off to the CPU. In Power Down (PD = 1), the oscillator is frozen. The Idle and Power Down modes are activated by setting bits in Special Function Register PCON. The address of this register is 87H. Figure 26 details its contents.

In the HMOS devices the PCON register only contains SMOD. The other four bits are implemented only in the CHMOS devices. User software should never write 1s to unimplemented bits, since they may be used in future MCS-51 products.

IDLE MODE

An instruction that sets PCON.0 causes that to be the last instruction executed before going into the Idle

mode. In the Idle mode, the internal clock signal is gated off to the CPU, but not to the Interrupt, Timer, and Serial Port functions. The CPU status is preserved in its entirety: the Stack Pointer, Program Counter, Program Status Word, Accumulator, and all other registers maintain their data during Idle. The port pins hold the logical states they had at the time Idle was activated. ALE and $\overline{PSEN}$ hold at logic high levels.

There are two ways to terminate the Idle. Activation of any enabled interrupt will cause PCON.0 to be cleared by hardware, terminating the Idle mode. The interrupt will be serviced, and following RETI the next instruction to be executed will be the one following the instruction that put the device into Idle.

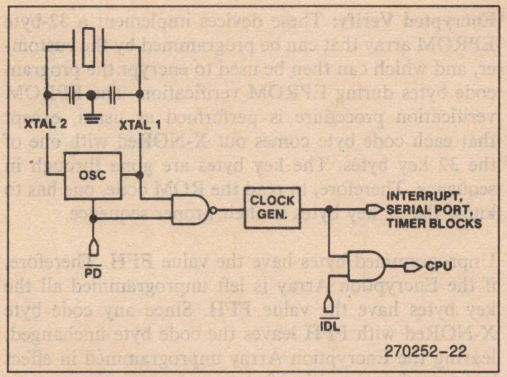


Figure 27. Idle and Power Down Hardware

(MSB)							(LSB)
SMOD	-	-	-	GF1	GF0	PD	IDL

Symbol	Position	Name and Function
SMOD	PCON.7	Double Baud rate bit. When set to a 1 and Timer 1 is used to generate baud rate, and the Serial Port is used in modes 1, 2, or 3.
—	PCON.6	(Reserved)
—	PCON.5	(Reserved)
—	PCON.4	(Reserved)
GF1	PCON.3	General-purpose flag bit.
GF0	PCON.2	General-purpose flag bit.
PD	PCON.1	Power Down bit. Setting this bit activates power down operation.
IDL	PCON.0	Idle mode bit. Setting this bit activates idle mode operation.

If 1s are written to PD and IDL at the same time, PD takes precedence. The reset value of PCON is (0XXX0000).
In the HMOS devices the PCON register only contains SMOD. The other four bits are implemented only in the CHMOS devices. User software should never write 1s to unimplemented bits, since they may be used in future MCS-51 products.

Figure 28. PCON: Power Control Register

The flag bits GF0 and GF1 can be used to give an indication if an interrupt occurred during normal operation or during an Idle. For example, an instruction that activates Idle can also set one or both flag bits. When Idle is terminated by an interrupt, the interrupt service routine can examine the flag bits.

The other way of terminating the Idle mode is with a hardware reset. Since the clock oscillator is still running, the hardware reset needs to be held active for only two machine cycles (24 oscillator periods) to complete the reset.

The signal at the RST pin clears the IDL bit directly and asynchronously. At this time the CPU resumes program execution from where it left off; that is, at the instruction following the one that invoked the Idle Mode. As shown in Figure 25, two or three machine cycles of program execution may take place before the internal reset algorithm takes control. On-chip hardware inhibits access to the internal RAM during this time, but access to the port pins is not inhibited. To eliminate the possibility of unexpected outputs at the port pins, the instruction following the one that invokes Idle should not be one that writes to a port pin or to external Data RAM.

POWER DOWN MODE

An instruction that sets PCON.1 causes that to be the last instruction executed before going into the Power Down mode. In the Power Down mode, the on-chip oscillator is stopped. With the clock frozen, all functions are stopped, but the on-chip RAM and Special Function Registers are held. The port pins output the values held by their respective SFRs. ALE and $\overline{PSEN}$ output lows.

The only exit from Power Down is a hardware reset. Reset redefines all the SFRs, but does not change the on-chip RAM.

In the Power Down mode of operation, VCC can be reduced to as low as 2V. Care must be taken, however, to ensure that VCC is not reduced before the Power Down mode is invoked, and that VCC is restored to its normal operating level, before the Power Down mode is terminated. The reset that terminates Power Down also frees the oscillator. The reset should not be activated before VCC is restored to its normal operating level, and must be held active long enough to allow the oscillator to restart and stabilize (normally less than 10 msec).

EPROM VERSIONS

The EPROM versions of these devices are listed in Table 4. The 8751H programs at VPP = 21V using one 50 msec $\overline{PROG}$ pulse per byte programmed. This results in a total programming time (4K bytes) of approximately 4 minutes.

Table 4. EPROM Versions of the 8051 and 8052

Device Name	EPROM Version	EPROM Bytes	Ckt Type	VPP	Time Required to Program Entire Array
8051	(8751)	4K	HMOS	21.0V	4 minutes
8051AH	8751H	4K	HMOS	21.0V	4 minutes
80C51BH	87C51	4K	CHMOS	12.75V	13 seconds
8052AH	8752BH	8K	HMOS	12.75V	26 seconds

The 8752BH and 87C51 use the faster "Quick-Pulse" programming™ algorithm. These devices program at $\overline{VPP}$ = 12.75V using a series of twenty-five 100 µs $\overline{PROG}$ pulses per byte programmed. This results in a total programming time of approximately 26 seconds for the 8752BH (8K bytes) and 13 seconds for the 87C51 (4K bytes).

Detailed procedures for programming and verifying each device are given in the data sheets.

EXPOSURE TO LIGHT

It is good practice to cover the EPROM window with an opaque label when the device is in operation. This is not so much to protect the EPROM array from inadvertent erasure, but to protect the RAM and other on-chip logic. Allowing light to impinge on the silicon die while the device is operating can cause logical malfunction.

Program Memory Locks

In some microcontroller applications it is desirable that the Program Memory be secure from software piracy. Intel has responded to this need by implementing a Program Memory locking scheme in some of the MCS-51 devices. While it is impossible for anyone to guarantee absolute security against all levels of technological sophistication, the Program Memory locks in the MCS-51 devices will present a formidable barrier against illegal readout of protected software.

8751H

The 8751H contains a lock bit which, once programmed, denies electrical access by any external means to the on-chip Program Memory. The effect of this lock bit is that while it is programmed the internal Program Memory can not be read out, the device can not be further programmed, and it *can not execute external Program Memory*. Erasing the EPROM array deactivates the lock bit and restores the device's full functionality. It can then be re-programmed.

The procedure for programming the lock bit is detailed in the 8751H data sheet.

87C51 AND 8752BH

The 87C51 and 8752BH contain two Program Memory locking schemes: Encrypted Verify and Lock Bits.

Encrypted Verify: These devices implement a 32-byte EPROM array that can be programmed by the customer, and which can then be used to encrypt the program code bytes during EPROM verification. The EPROM verification procedure is performed as usual, except that each code byte comes out X-NORed with one of the 32 key bytes. The key bytes are gone through in sequence. Therefore, to read the ROM code, one has to know the 32 key bytes in their proper sequence.

Unprogrammed bytes have the value FFH. Therefore, if the Encryption Array is left unprogrammed all the key bytes have the value FFH. Since any code byte X-NORed with FFH leaves the code byte unchanged, leaving the Encryption Array unprogrammed in effect bypasses the encryption feature.

Lock Bits: Also on the chip are two Lock Bits which can be left unprogrammed (U) or programmed (P) to obtain the following features:

Bit 2	Bit 1	Additional Features
U	U	None
U	P	• Externally fetched code can not access internal Program Memory. • Further programming disabled.
P	U	(Reserved for Future definition.)
P	P	• Externally fetched code can not access internal Program Memory. • Further programming disabled. • Program verification is disabled.

When Lock Bit 1 is programmed, the logic level at the $\overline{EA}$ pin is sampled and latched during reset. If the device is powered up without a reset, the latch initializes to a random value, and holds that value until reset is activated. It is necessary that the latched value of $\overline{EA}$ be in agreement with the current logic level at that pin in order for the device to function properly.

ONCE Mode in the 87C51

The ONCE ("on-circuit emulation") mode facilitates testing and debugging of systems using the 87C51 without the 87C51 having to be removed from the circuit. The ONCE mode is invoked by:

1. Pull ALE low while the device is in reset and $\overline{PSEN}$ is high;
2. Hold ALE low as RST is deactivated.

While the device is in ONCE mode, the Port 0 pins go into a float state, and the other port pins and ALE and $\overline{PSEN}$ are weakly pulled high. The oscillator circuit remains active. While the 87C51 is in this mode, an emulator or test CPU can be used to drive the circuit. Normal operation is restored after a normal reset is applied.

THE ON-CHIP OSCILLATORS

HMOS Versions

The on-chip oscillator circuitry for the HMOS (HMOS-I and HMOS-II) members of the MCS-51 family is a single stage linear inverter (Figure 29), intended for use as a crystal-controlled, positive reactance oscillator (Figure 30). In this application the crystal is operated in its fundamental response mode as an inductive reactance in parallel resonance with capacitance external to the crystal.

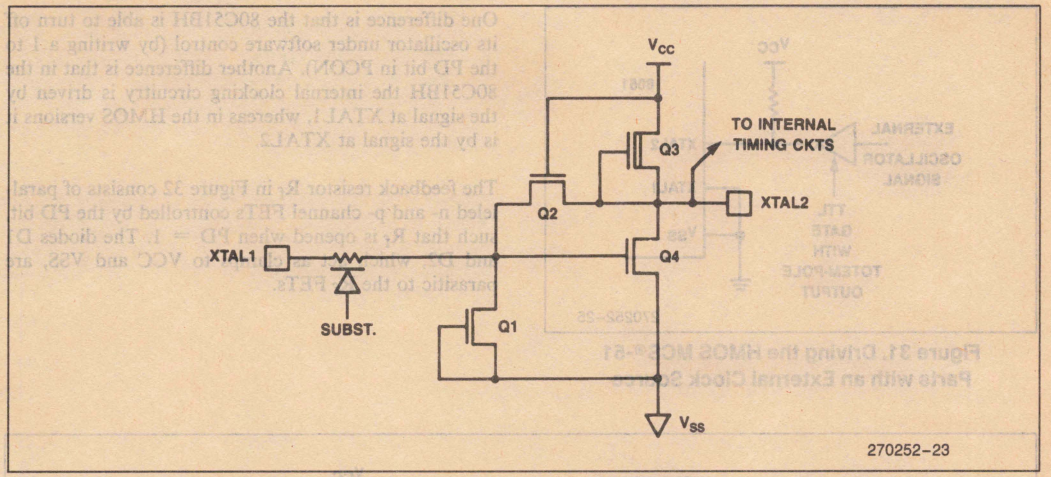


Figure 29. On-Chip Oscillator Circuitry in the HMOS Versions of the MCS®-51 Family

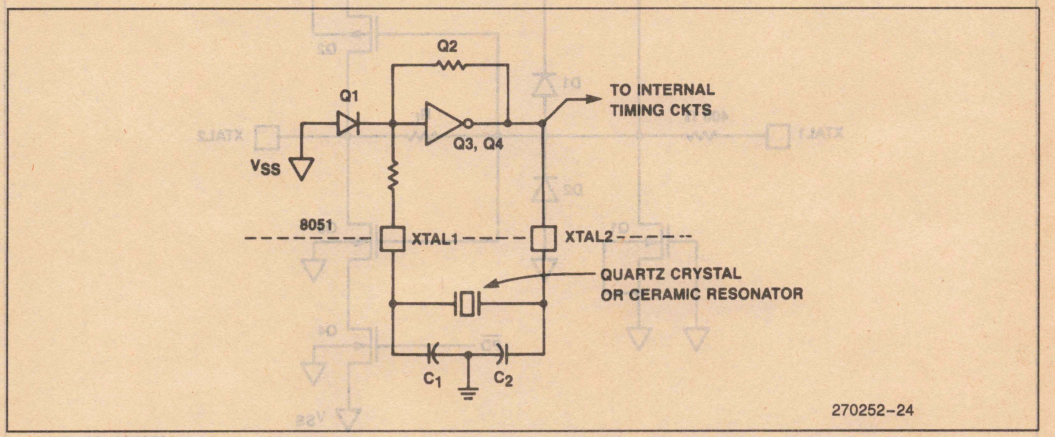


Figure 30. Using the HMOS On-Chip Oscillator

The crystal specifications and capacitance values (C1 and C2 in Figure 30) are not critical. 30 pF can be used in these positions at any frequency with good quality crystals. A ceramic resonator can be used in place of the crystal in cost-sensitive applications. When a ceramic resonator is used, C1 and C2 are normally selected to be of somewhat higher values, typically, 47 pF. The manufacturer of the ceramic resonator should be consulted for recommendations on the values of these capacitors.

A more in-depth discussion of crystal specifications, ceramic resonators, and the selection of values for C1 and C2 can be found in Application Note AP-155, "Oscillators for Microcontrollers," which is included in this manual.

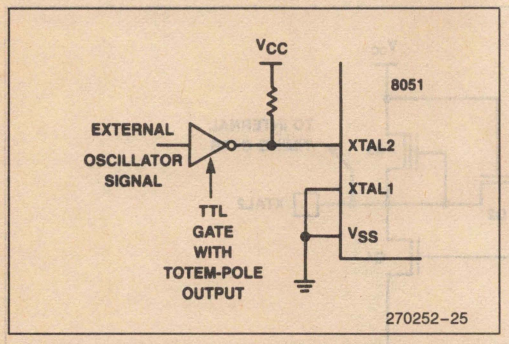


Figure 31. Driving the HMOS MCS®-51 Parts with an External Clock Source

To drive the HMOS parts with an external clock source, apply the external clock signal to XTAL2, and ground XTAL1, as shown in Figure 31. A pullup resistor may be used (to increase noise margin), but is optional if VOH of the driving gate exceeds the VIH MIN specification of XTAL2.

CHMOS VERSIONS

The on-chip oscillator circuitry for the 80C51BH, shown in Figure 32, consists of a single stage linear inverter intended for use as a crystal-controlled, positive reactance oscillator in the same manner as the HMOS parts. However, there are some important differences.

One difference is that the 80C51BH is able to turn off its oscillator under software control (by writing a 1 to the PD bit in PCON). Another difference is that in the 80C51BH the internal clocking circuitry is driven by the signal at XTAL1, whereas in the HMOS versions it is by the signal at XTAL2.

The feedback resistor R_f in Figure 32 consists of paralleled n- and p- channel FETs controlled by the PD bit, such that R_f is opened when PD = 1. The diodes D1 and D2, which act as clamps to VCC and VSS, are parasitic to the R_f FETs.

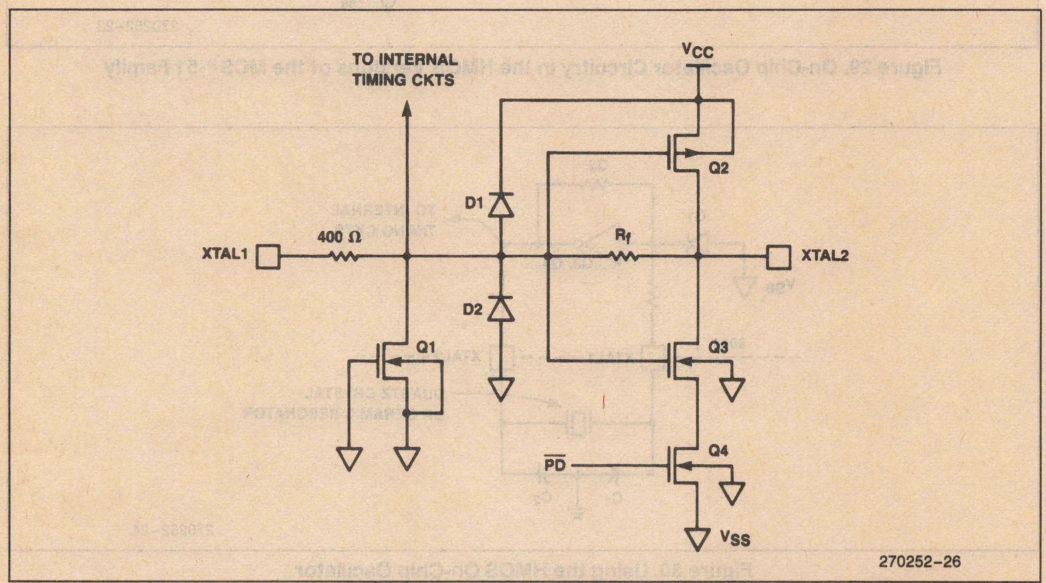


Figure 32. On-Chip Oscillator Circuitry in the CHMOS Versions of the MCS®-51 Family

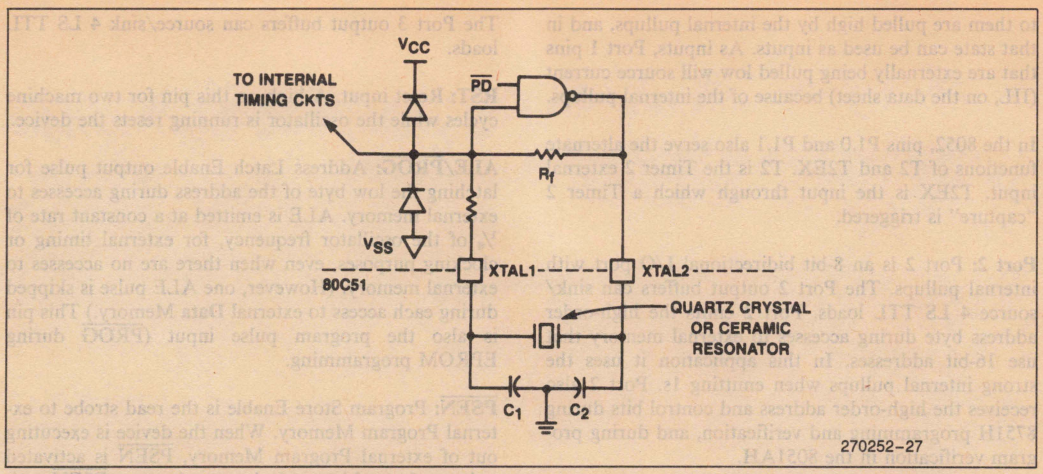


Figure 33. Using the CHMOS On-Chip Oscillator

The oscillator can be used with the same external components as the HMOS versions, as shown in Figure 33. Typically, C1 = C2 = 30 pF when the feedback element is a quartz crystal, and C1 = C2 = 47 pF when a ceramic resonator is used.

To drive the CHMOS parts with an external clock source, apply the external clock signal to XTAL1, and leave XTAL2 float, as shown in Figure 34.

The reason for this change from the way the HMOS part is driven can be seen by comparing Figures 29 and 32. In the HMOS devices the internal timing circuits are driven by the signal at XTAL2. In the CHMOS devices the internal timing circuits are driven by the signal at XTAL1.

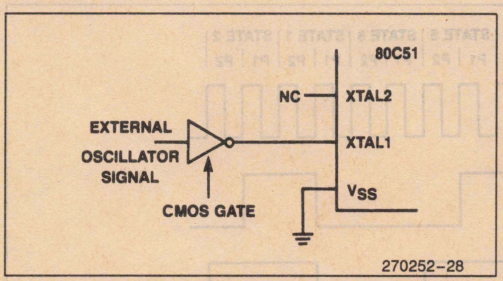


Figure 34. Driving the CHMOS MCS®-51 Parts with an External Clock Source

INTERNAL TIMING

Figures 35 through 38 show when the various strobe and port signals are clocked internally. The figures do not show rise and fall times of the signals, nor do they show propagation delays between the XTAL2 signal and events at other pins.

Rise and fall times are dependent on the external loading that each pin must drive. They are often taken to be something in the neighborhood of 10 nsec, measured between 0.8V and 2.0V.

Propagation delays are different for different pins. For a given pin they vary with pin loading, temperature, VCC, and manufacturing lot. If the XTAL2 waveform is taken as the timing reference, prop delays may vary from 25 to 125 nsec.

The AC Timings section of the data sheets do not reference any timing to the XTAL2 waveform. Rather, they relate the critical edges of control and input signals to each other. The timings published in the data sheets include the effects of propagation delays under the specified test conditions.

MCS®-51 PIN DESCRIPTIONS

VCC: Supply voltage.

VSS: Circuit ground potential.

Port 0: Port 0 is an 8-bit open drain bidirectional I/O port. As an open drain output port it can sink 8 LS TTL loads. Port 0 pins that have 1s written to them float, and in that state will function as high-impedance imputs. Port 0 is also the multiplexed low-order address and data bus during accesses to external memory. In this application it uses strong internal pullups when emitting 1s. Port 0 also emits code bytes during program verification. In that application, external pullups are required.

Port 1: Port 1 is an 8-bit bidirectional I/O port with internal pullups. The Port 1 output buffers can sink/source 4 LS TTL loads. Port 1 pins that have 1s written

to them are pulled high by the internal pullups, and in that state can be used as inputs. As inputs, Port 1 pins that are externally being pulled low will source current (IIL, on the data sheet) because of the internal pullups.

In the 8052, pins P1.0 and P1.1 also serve the alternate functions of T2 and T2EX. T2 is the Timer 2 external input. T2EX is the input through which a Timer 2 "capture" is triggered.

Port 2: Port 2 is an 8-bit bidirectional I/O port with internal pullups. The Port 2 output buffers can sink/source 4 LS TTL loads. Port 2 emits the high-order address byte during accesses to external memory that use 16-bit addresses. In this application it uses the strong internal pullups when emitting 1s. Port 2 also receives the high-order address and control bits during 8751H programming and verification, and during program verification in the 8051AH.

Port 3: Port 3 is an 8-bit bidirectional I/O port with internal pullups. It also serves the functions of various special features of the MCS-51 Family, as listed below:

Port Pin	Alternate Function
P3.0	RXD (serial input port)
P3.1	TXD (serial output port)
P3.2	$\overline{INT0}$ (external interrupt 0)
P3.3	$\overline{INT1}$ (external interrupt 1)
P3.4	T0 (Timer 0 external input)
P3.5	T1 (Timer 1 external input)
P3.6	$\overline{WR}$ (external data memory write strobe)
P3.7	$\overline{RD}$ (external data memory read strobe)

The Port 3 output buffers can source/sink 4 LS TTL loads.

RST: Reset input. A high on this pin for two machine cycles while the oscillator is running resets the device.

ALE/$\overline{PROG}$: Address Latch Enable output pulse for latching the low byte of the address during accesses to external memory. ALE is emitted at a constant rate of $\frac{1}{6}$ of the oscillator frequency, for external timing or clocking purposes, even when there are no accesses to external memory. (However, one ALE pulse is skipped during each access to external **Data** Memory.) This pin is also the program pulse input ($\overline{PROG}$ during EPROM programming.

$\overline{PSEN}$: Program Store Enable is the read strobe to external Program Memory. When the device is executing out of external Program Memory, $\overline{PSEN}$ is activated twice each machine cycle (except that two $\overline{PSEN}$ activations are skipped during accesses to external **Data** Memory). $\overline{PSEN}$ is not activated when the device is executing out of internal Program Memory.

$\overline{EA}$/VPP: When $\overline{EA}$ is held high the CPU executes out of internal Program Memory (unless the Program Counter exceeds 0FFFH in the 8051AH, or 1FFFH in the 8052). Holding $\overline{EA}$ low forces the CPU to execute out of external memory regardless of the Program Counter value. In the 8031AH and 8032, $\overline{EA}$ must be extremely wired low. In the EPROM devices, this pin also receives the programming supply voltage (VPP) during EPROM programming.

XTAL1: Input to the inverting oscillator amplifier.

XTAL2: Output from the inverting oscillator amplifier.

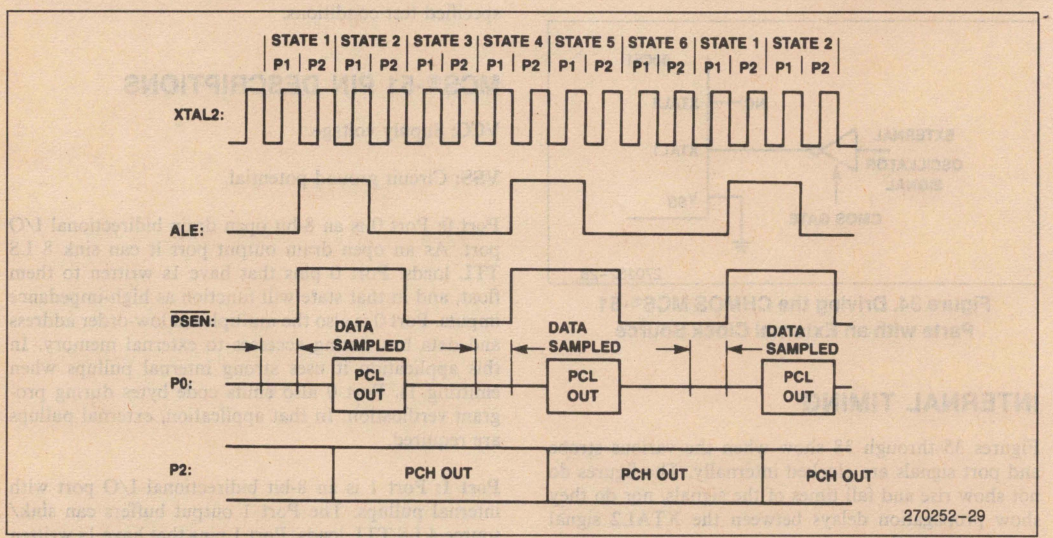


Figure 35. External Program Memory Fetches

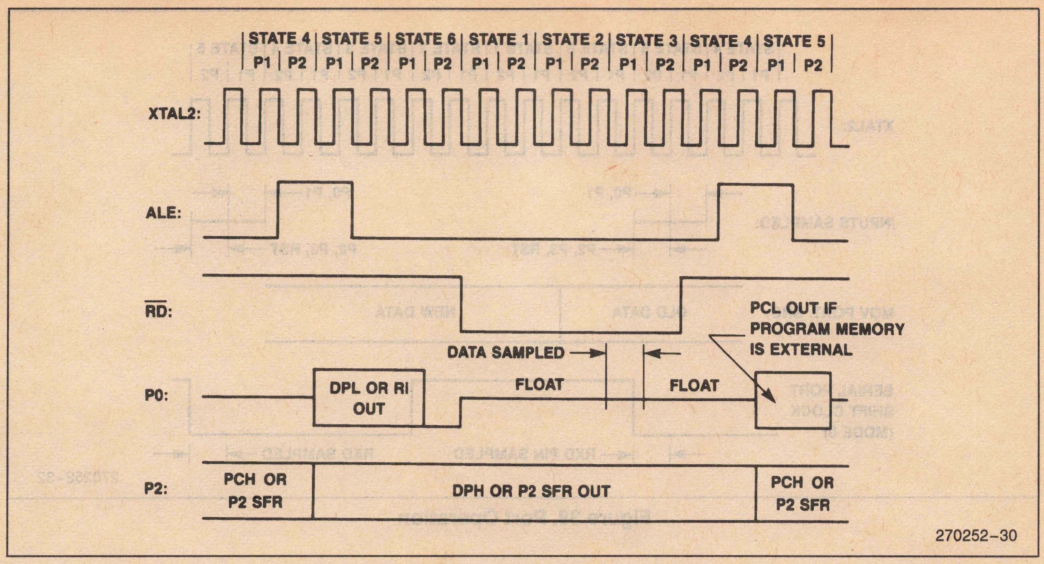


Figure 36. External Data Memory Read Cycle

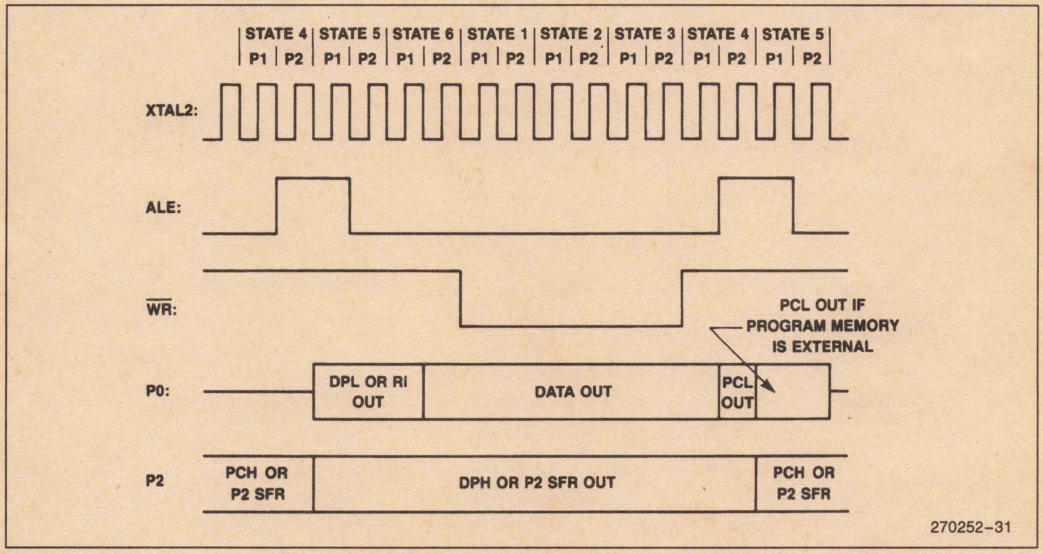


Figure 37. External Data Memory Write Cycle

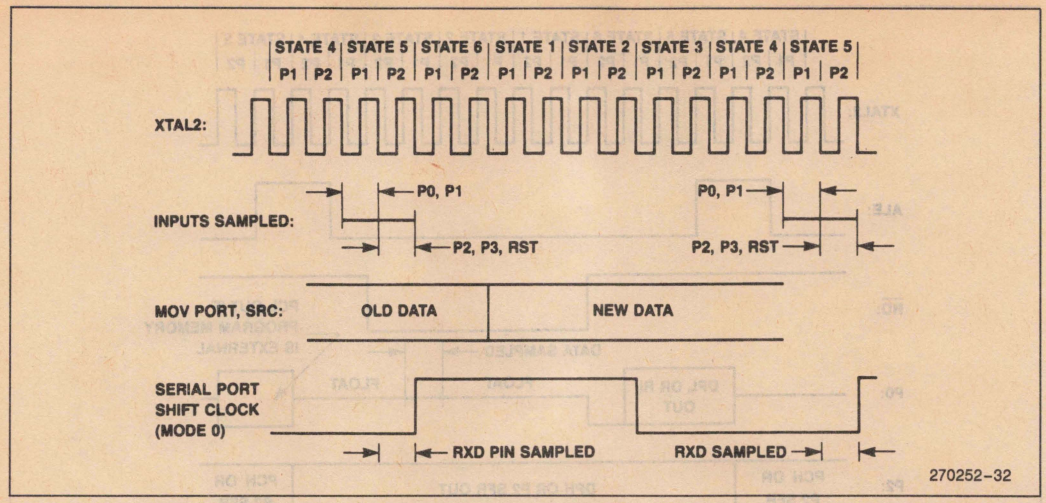


Figure 38. Port Operation

MCS®-51 Programmers Guide and Instruction Set

7

7 MCS®-51 Programmers Guide and Instruction Set

MCS®-51 PROGRAMMER'S GUIDE AND INSTRUCTION SET

The information presented in this chapter is collected from the previous MCS®-51 chapters of this book. The material has been selected and rearranged to form a quick and convenient reference for the programmers of the MCS-51.

The following list should make it easier to find a subject in this chapter.

Memory Organization
- **Program Memory** 7-2
- **Data Memory** 7-3
- **Direct and Indirect Address Area** 7-5

Special Function Registers 7-7
Contents of SFRs after Power-On 7-8
SFR Memory Map 7-9
Program Status Word (PSW) 7-10
Power Control Register (PCON) 7-10
Interrupts 7-11
Interrupt Enable Register (IE) 7-11
Assigning Priority Level 7-12
Interrupt Priority Register 7-12
Timer/Counter Control Register (TCON) 7-13
Timer/Counter Mode Control Register (TMOD) 7-13
Timer Set-Up 7-14
- **Timer/Counter 0** 7-14
- **Timer/Counter 1** 7-15

Timer/Counter 2 Control Register (T2CON) 7-16
Timer/Counter 2 Set-Up 7-17
Serial Port Control Register 7-18
Serial Port Set-Up 7-18
Generating Baud Rates 7-19
MCS-51 Instruction Set 7-20
Instruction Definitions 7-24

MEMORY ORGANIZATION

PROGRAM MEMORY

The 8051 has separate address spaces for Program Memory and Data Memory. The Program Memory can be up to 64K bytes long. The lower 4K (8K for the 8052) may reside on-chip.

Figure 1 shows a map of the 8051 program memory, and Figure 2 shows a map of the 8052 program memory.

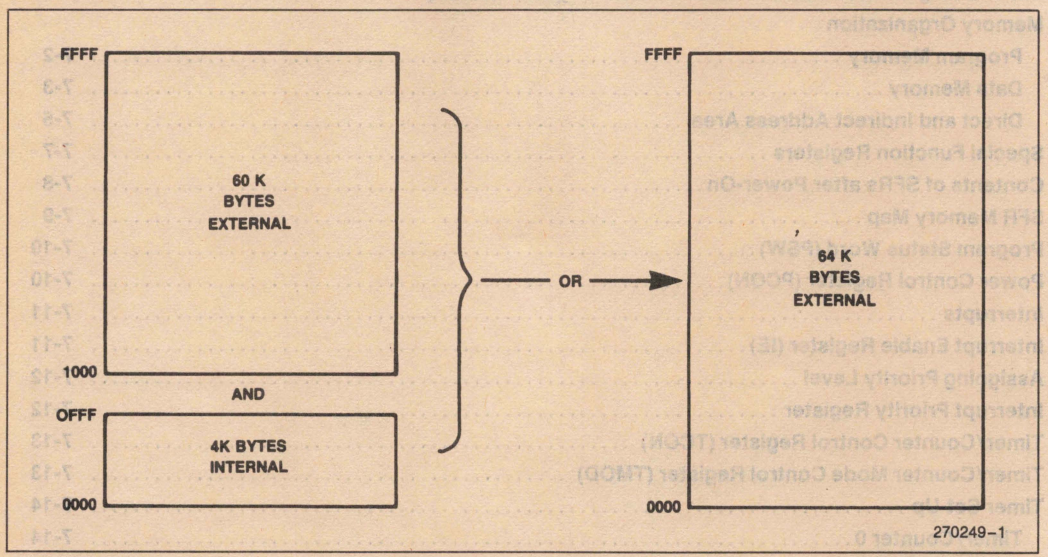


Figure 1. The 8051 Program Memory

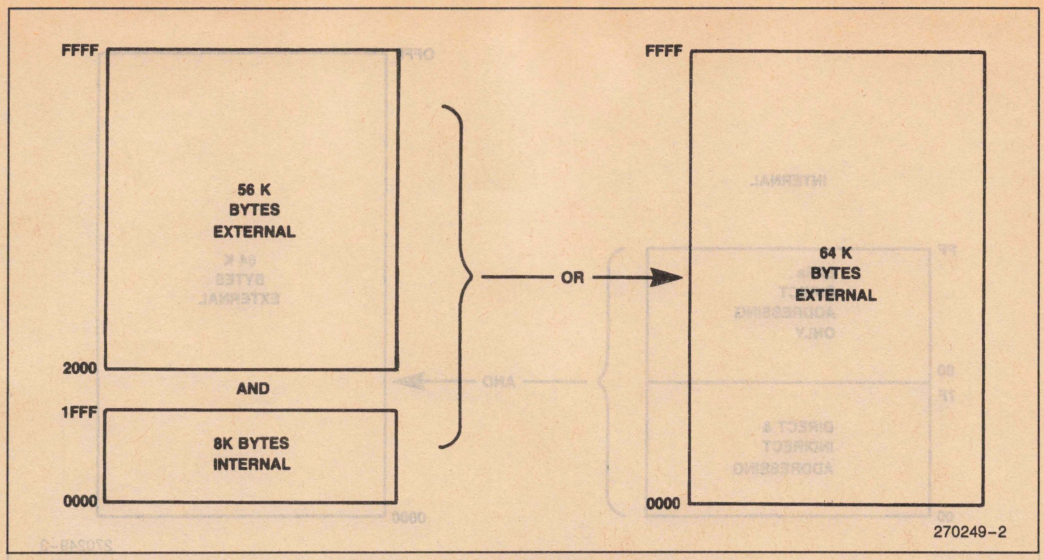


Figure 2. The 8052 Program Memory

Data Memory:

The 8051 can address up to 64K bytes of Data Memory to the chip. The "MOVX" instruction is used to access the external data memory. (Refer to the MCS-51 Instruction Set, in this chapter, for detailed description of instructions).

The 8051 has 128 bytes of on-chip RAM (256 bytes in the 8052) plus a number of Special Function Registers (SFRs). The lower 128 bytes of RAM can be accessed either by direct addressing (MOV data addr) or by indirect addressing (MOV @Ri). Figure 3 shows the 8051 and the 8052 Data Memory organization.

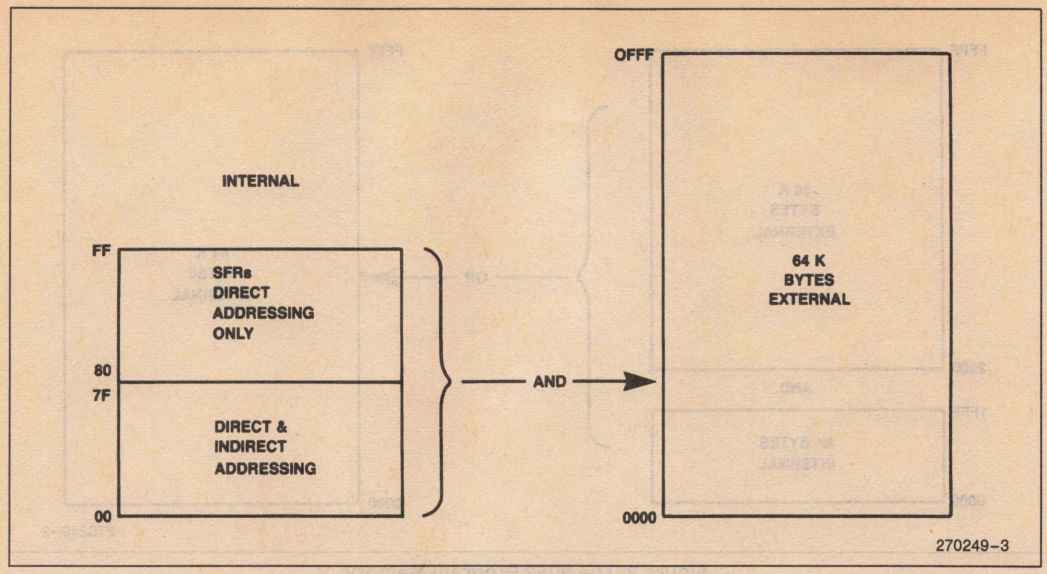


Figure 3a. The 8051 Data Memory

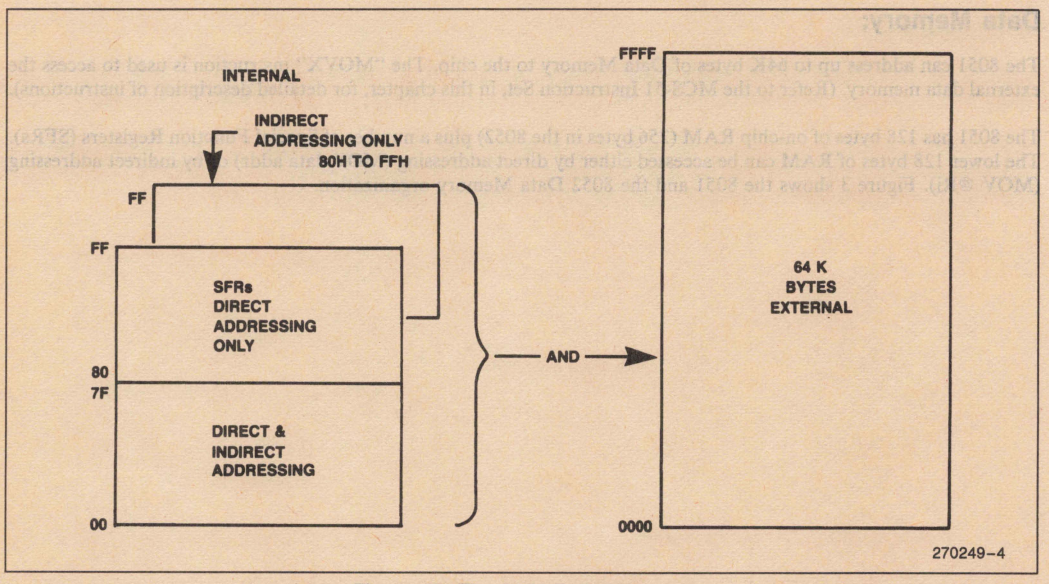


Figure 3b. The 8052 Data Memory

INDIRECT ADDRESS AREA:

Note that in Figure 3b the SFRs and the indirect address RAM have the same addresses (80H–0FFH). Nevertheless, they are two separate areas and are accessed in two different ways.

For example the instruction

```
MOV    80H,#0AAH
```

writes 0AAH to Port 0 which is one of the SFRs and the instruction

```
MOV    R0,#80H

MOV    @R0,#0BBH
```

writes 0BBH in location 80H of the data RAM. Thus, after execution of both of the above instructions Port 0 will contain 0AAH and location 80 of the RAM will contain 0BBH.

DIRECT AND INDIRECT ADDRESS AREA:

The 128 bytes of RAM which can be accessed by both direct and indirect addressing can be divided into 3 segments as listed below and shown in Figure 4.

1. Register Banks 0-3: Locations 0 through 1FH (32 bytes). ASM-51 and the device after reset default to register bank 0. To use the other register banks the user must select them in the software (refer to the MCS-51 Micro Assembler User's Guide). Each register bank contains 8 one-byte registers, 0 through 7.

Reset initializes the Stack Pointer to location 07H and it is incremented once to start from location 08H which is the first register (RO) of the second register bank. Thus, in order to use more than one register bank, the SP should be intialized to a different location of the RAM where it is not used for data storage (ie, higher part of the RAM).

2. Bit Addressable Area: 16 bytes have been assigned for this segment, 20H-2FH. Each one of the 128 bits of this segment can be directly addressed (0-7FH).

The bits can be referred to in two ways both of which are acceptable by the ASM-51. One way is to refer to their addresses, ie. 0 to 7FH. The other way is with reference to bytes 20H to 2FH. Thus, bits 0–7 can also be referred to as bits 20.0–20.7, and bits 8-FH are the same as 21.0–21.7 and so on.

Each of the 16 bytes in this segment can also be addressed as a byte.

3. Scratch Pad Area: Bytes 30H through 7FH are available to the user as data RAM. However, if the stack pointer has been initialized to this area, enough number of bytes should be left aside to prevent SP data destruction.

Figure 4 shows the different segments of the on-chip RAM.

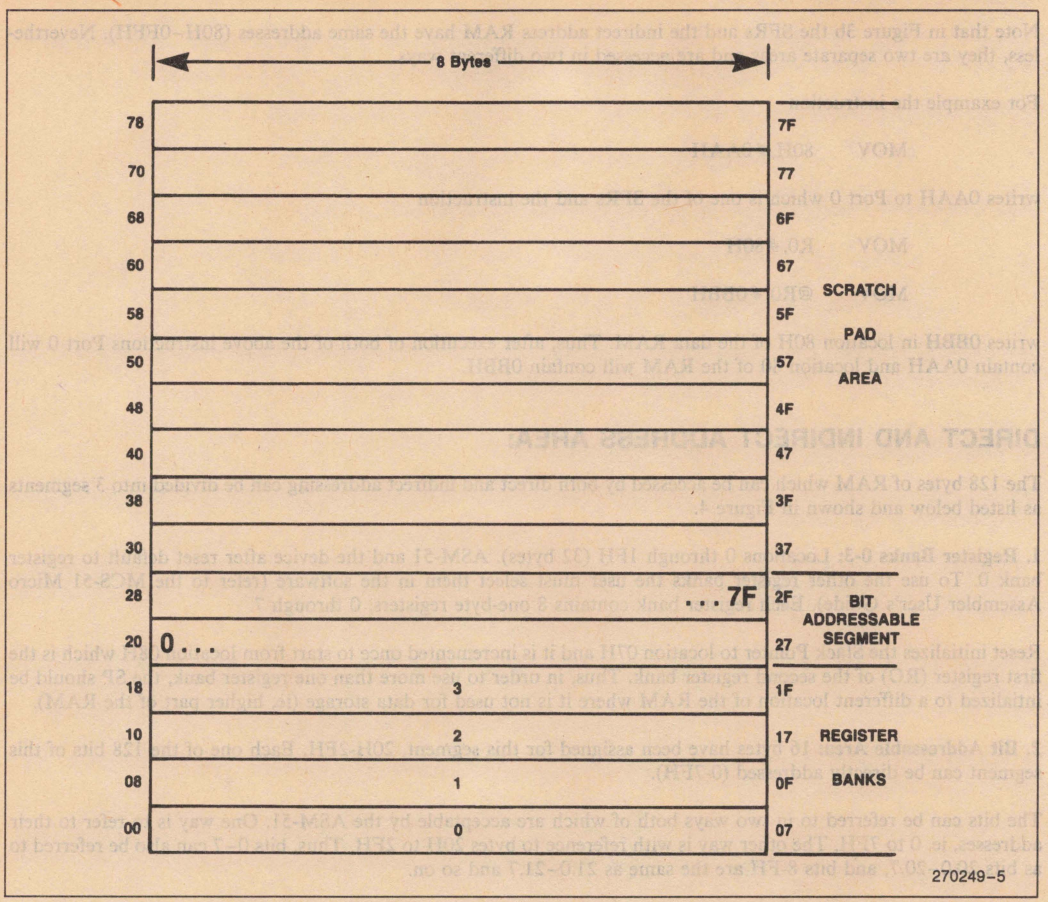


Figure 4. 128 Bytes of RAM Direct and Indirect Addressable

SPECIAL FUNCTION REGISTERS:

Table 1 contains a list of all the SFRs and their addresses.

Comparing Table 1 and Figure 5 shows that all of the SFRs that are byte and bit addressable are located on the first column of the diagram in Figure 5.

Table 1

Symbol	Name	Address
*ACC	Accumulator	0E0H
*B	B Register	0F0H
*PSW	Program Status Word	0D0H
SP	Stack Pointer	81H
DPTR	Data Pointer 2 Bytes	
DPL	Low Byte	82H
DPH	High Byte	83H
*P0	Port 0	80H
*P1	Port 1	90H
*P2	Port 2	0A0H
*P3	Port 3	0B0H
*IP	Interrupt Priority Control	0B8H
*IE	Interrupt Enable Control	0A8H
TMOD	Timer/Counter Mode Control	89H
*TCON	Timer/Counter Control	88H
*+T2CON	Timer/Counter 2 Control	0C8H
TH0	Timer/Counter 0 High Byte	8CH
TL0	Timer/Counter 0 Low Byte	8AH
TH1	Timer/Counter 1 High Byte	8DH
TL1	Timer/Counter 1 Low Byte	8BH
+TH2	Timer/Counter 2 High Byte	0CDH
+TL2	Timer/Counter 2 Low Byte	0CCH
+RCAP2H	T/C 2 Capture Reg. High Byte	0CBH
+RCAP2L	T/C 2 Capture Reg. Low Byte	0CAH
*SCON	Serial Control	98H
SBUF	Serial Data Buffer	99H
PCON	Power Control	87H

* = Bit addressable
+ = 8052 only

WHAT DO THE SFRs CONTAIN JUST AFTER POWER-ON OR A RESET?

Table 2 lists the contents of each SFR after power-on or a hardware reset.

Table 2. Contents of the SFRs after reset

Register	Value in Binary
*ACC	00000000
*B	00000000
*PSW	00000000
SP	00000111
DPTR	
DPH	00000000
DPL	00000000
*P0	11111111
*P1	11111111
*P2	11111111
*P3	11111111
*IP	8051 XXX00000, 8052 XX000000
*IE	8051 0XX00000, 8052 0X000000
TMOD	00000000
*TCON	00000000
*+T2CON	00000000
TH0	00000000
TL0	00000000
TH1	00000000
TL1	00000000
+TH2	00000000
+TL2	00000000
+RCAP2H	00000000
+RCAP2L	00000000
*SCON	00000000
SBUF	Indeterminate
PCON	HMOS 0XXXXXXX CHMOS 0XXX0000

X = Undefined
* = Bit Addressable
\+ = 8052 only

SFR MEMORY MAP

	8 Bytes							
F8								FF
F0	B							F7
E8								EF
E0	ACC							E7
D8								DF
D0	PSW							D7
C8	T2CON		RCAP2L	RCAP2H	TL2	TH2		CF
C0								C7
B8	IP							BF
B0	P3							B7
A8	IE							AF
A0	P2							A7
98	SCON	SBUF						9F
90	P1							97
88	TCON	TMOD	TL0	TL1	TH0	TH1		8F
80	P0	SP	DPL	DPH			PCON	87

↑ Bit Addressable

Figure 5

Those SFRs that have their bits assigned for various functions are listed in this section. A brief description of each bit is provided for quick reference. For more detailed information refer to the Architecture Chapter of this book.

PSW: PROGRAM STATUS WORD. BIT ADDRESSABLE.

CY	AC	F0	RS1	RS0	OV	—	P

CY	PSW.7	Carry Flag.
AC	PSW.6	Auxiliary Carry Flag.
F0	PSW.5	Flag 0 available to the user for general purpose.
RS1	PSW.4	Register Bank selector bit 1 (SEE NOTE 1).
RS0	PSW.3	Register Bank selector bit 0 (SEE NOTE 1).
OV	PSW.2	Overflow Flag.
—	PSW.1	Not implemented, reserved for future use.*
P	PSW.0	Parity flag. Set/cleared by hardware each instruction cycle to indicate an odd/even number of '1' bits in the accumulator.

NOTE:
1. The value presented by RS0 and RS1 selects the corresponding register bank.

RS1	RS0	Register Bank	Address
0	0	0	00H-07H
0	1	1	08H-0FH
1	0	2	10H-17H
1	1	3	18H-1FH

*User software should not write 1s to reserved bits. These bits may be used in future MCS-51 products to invoke new features. In that case, the reset or inactive value of the new bit will be 0, and its active value will be 1.

PCON: POWER CONTROL REGISTER. NOT BIT ADDRESSABLE.

SMOD	—	—	—	GF1	GF0	PD	IDL

SMOD	Double baud rate bit. If Timer 1 is used to generate baud rate and SMOD = 1, the baud rate is doubled when the Serial Port is used in modes 1, 2, or 3.
—	Not implemented, reserved for future use.*
—	Not implemented, reserved for future use.*
—	Not implemented, reserved for future use.*
GF1	General purpose flag bit.
GF0	General purpose flag bit.
PD	Power Down bit. Setting this bit activates Power Down operation in the 80C51BH. (Available only in CHMOS).
IDL	Idle Mode bit. Setting this bit activates Idle Mode operation in the 80C51BH. (Available only in CHMOS).

If 1s are written to PD and IDL at the same time, PD takes precedence.

*User software should not write 1s to reserved bits. These bits may be used in future MCS-51 products to invoke new features. In that case, the reset or inactive value of the new bit will be 0, and its active value will be 1.

INTERRUPTS:

In order to use any of the interrupts in the MCS-51, the following three steps must be taken.

1. Set the EA (enable all) bit in the IE register to 1.
2. Set the corresponding individual interrupt enable bit in the IE register to 1.
3. Begin the interrupt service routine at the corresponding Vector Address of that interrupt. See Table below.

Interrupt Source	Vector Address
IE0	0003H
TF0	000BH
IE1	0013H
TF1	001BH
RI & TI	0023H
TF2 & EXF2	002BH

In addition, for external interrupts, pins $\overline{INT0}$ and $\overline{INT1}$ (P3.2 and P3.3) must be set to 1, and depending on whether the interrupt is to be level or transition activated, bits IT0 or IT1 in the TCON register may need to be set to 1.

ITx = 0 level activated

ITx = 1 transition activated

IE: INTERRUPT ENABLE REGISTER. BIT ADDRESSABLE.

If the bit is 0, the corresponding interrupt is disabled. If the bit is 1, the corresponding interrupt is enabled.

EA	—	ET2	ES	ET1	EX1	ET0	EX0

EA	IE.7	Disables all interrupts. If EA = 0, no interrupt will be acknowledged. If EA = 1, each interrupt source is individually enabled or disabled by setting or clearing its enable bit.
—	IE.6	Not implemented, reserved for future use.*
ET2	IE.5	Enable or disable the Timer 2 overflow or capture interrupt (8052 only).
ES	IE.4	Enable or disable the serial port interrupt.
ET1	IE.3	Enable or disable the Timer 1 overflow interrupt.
EX1	IE.2	Enable or disable External Interrupt 1.
ET0	IE.1	Enable or disable the Timer 0 overflow interrupt.
EX0	IE.0	Enable or disable External Interrupt 0.

*User software should not write 1s to reserved bits. These bits may be used in future MCS-51 products to invoke new features. In that case, the reset or inactive value of the new bit will be 0, and its active value will be 1.

ASSIGNING HIGHER PRIORITY TO ONE OR MORE INTERRUPTS:

In order to assign higher priority to an interrupt the corresponding bit in the IP register must be set to 1.

Remember that while an interrupt service is in progress, it cannot be interrupted by a lower or same level interrupt.

PRIORITY WITHIN LEVEL:

Priority within level is only to resolve simultaneous requests of the same priority level.

From high to low, interrupt sources are listed below:

IE0
TF0
IE1
TF1
RI or TI
TF2 or EXF2

IP: INTERRUPT PRIORITY REGISTER. BIT ADDRESSABLE.

If the bit is 0, the corresponding interrupt has a lower priority and if the bit is 1 the corresponding interrupt has a higher priority.

—	—	PT2	PS	PT1	PX1	PT0	PX0

—	IP. 7	Not implemented, reserved for future use.*
—	IP. 6	Not implemented, reserved for future use.*
PT2	IP. 5	Defines the Timer 2 interrupt priority level (8052 only).
PS	IP. 4	Defines the Serial Port interrupt priority level.
PT1	IP. 3	Defines the Timer 1 interrupt priority level.
PX1	IP. 2	Defines External Interrupt 1 priority level.
PT0	IP. 1	Defines the Timer 0 interrupt priority level.
PX0	IP. 0	Defines the External Interrupt 0 priority level.

*User software should not write 1s to reserved bits. These bits may be used in future MCS-51 products to invoke new features. In that case, the reset or inactive value of the new bit will be 0, and its active value will be 1.

TCON: TIMER/COUNTER CONTROL REGISTER. BIT ADDRESSABLE.

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

TF1 TCON. 7 Timer 1 overflow flag. Set by hardware when the Timer/Counter 1 overflows. Cleared by hardware as processor vectors to the interrupt service routine.

TR1 TCON. 6 Timer 1 run control bit. Set/cleared by software to turn Timer/Counter 1 ON/OFF.

TF0 TCON. 5 Timer 0 overflow flag. Set by hardware when the Timer/Counter 0 overflows. Cleared by hardware as processor vectors to the service routine.

TR0 TCON. 4 Timer 0 run control bit. Set/cleared by software to turn Timer/Counter 0 ON/OFF.

IE1 TCON. 3 External Interrupt 1 edge flag. Set by hardware when External Interrupt edge is detected. Cleared by hardware when interrupt is processed.

IT1 TCON. 2 Interrupt 1 type control bit. Set/cleared by software to specify falling edge/low level triggered External Interrupt.

IE0 TCON. 1 External Interrupt 0 edge flag. Set by hardware when External Interrupt edge detected. Cleared by hardware when interrupt is processed.

IT0 TCON. 0 Interrupt 0 type control bit. Set/cleared by software to specify falling edge/low level triggered External Interrupt.

TMOD: TIMER/COUNTER MODE CONTROL REGISTER. NOT BIT ADDRESSABLE.

GATE	C/$\overline{T}$	M1	M0	GATE	C/$\overline{T}$	M1	M0
TIMER 1				TIMER 0			

GATE When TRx (in TCON) is set and GATE = 1, TIMER/COUNTERx will run only while INTx pin is high (hardware control). When GATE = 0, TIMER/COUNTERx will run only while TRx = 1 (software control).

C/$\overline{T}$ Timer or Counter selector. Cleared for Timer operation (input from internal system clock). Set for Counter operation (input from Tx input pin).

M1 Mode selector bit. (NOTE 1)

M0 Mode selector bit. (NOTE 1)

NOTE 1:

M1	M0		Operating Mode
0	0	0	13-bit Timer (MCS-48 compatible)
0	1	1	16-bit Timer/Counter
1	0	2	8-bit Auto-Reload Timer/Counter
1	1	3	(Timer 0) TL0 is an 8-bit Timer/Counter controlled by the standard Timer 0 control bits, TH0 is an 8-bit Timer and is controlled by Timer 1 control bits.
1	1	3	(Timer 1) Timer/Counter 1 stopped.

TIMER SET-UP

Tables 3 through 6 give some values for TMOD which can be used to set up Timer 0 in different modes.

It is assumed that only one timer is being used at a time. If it is desired to run Timers 0 and 1 simultaneously, in any mode, the value in TMOD for Timer 0 must be ORed with the value shown for Timer 1 (Tables 5 and 6).

For example, if it is desired to run Timer 0 in mode 1 GATE (external control), and Timer 1 in mode 2 COUNTER, then the value that must be loaded into TMOD is 69H (09H from Table 3 ORed with 60H from Table 6).

Moreover, it is assumed that the user, at this point, is not ready to turn the timers on and will do that at a different point in the program by setting bit TRx (in TCON) to 1.

TIMER/COUNTER 0

As a Timer:

Table 3

MODE	TIMER 0 FUNCTION	TMOD	
		INTERNAL CONTROL (NOTE 1)	EXTERNAL CONTROL (NOTE 2)
0	13-bit Timer	00H	08H
1	16-bit Timer	01H	09H
2	8-bit Auto-Reload	02H	0AH
3	two 8-bit Timers	03H	0BH

As a Counter:

Table 4

MODE	COUNTER 0 FUNCTION	TMOD	
		INTERNAL CONTROL (NOTE 1)	EXTERNAL CONTROL (NOTE 2)
0	13-bit Timer	04H	0CH
1	16-bit Timer	05H	0DH
2	8-bit Auto-Reload	06H	0EH
3	one 8-bit Counter	07H	0FH

NOTES:
1. The Timer is turned ON/OFF by setting/clearing bit TR0 in the software.
2. The Timer is turned ON/OFF by the 1 to 0 transition on $\overline{INT0}$ (P3.2) when TR0 = 1 (hardware control).

TIMER/COUNTER 1

As a Timer:

Table 5

MODE	TIMER 1 FUNCTION	TMOD	
		INTERNAL CONTROL (NOTE 1)	EXTERNAL CONTROL (NOTE 2)
0	13-bit Timer	00H	80H
1	16-bit Timer	10H	90H
2	8-bit Auto-Reload	20H	A0H
3	does not run	30H	B0H

As a Counter:

Table 6

MODE	COUNTER 1 FUNCTION	TMOD	
		INTERNAL CONTROL (NOTE 1)	EXTERNAL CONTROL (NOTE 2)
0	13-bit Timer	40H	C0H
1	16-bit Timer	50H	D0H
2	8-bit Auto-Reload	60H	E0H
3	not available	—	—

NOTES:
1. The Timer is turned ON/OFF by setting/clearing bit TR1 in the software.
2. The Timer is turned ON/OFF by the 1 to 0 transition on $\overline{INT1}$ (P3.3) when TR1 = 1 (hardware control).

T2CON: TIMER/COUNTER 2 CONTROL REGISTER. BIT ADDRESSABLE

8052 Only

TF2	EXF2	RCLK	TCLK	EXEN2	TR2	C/$\overline{T2}$	CP/$\overline{RL2}$

TF2 T2CON. 7 Timer 2 overflow flag set by hardware and cleared by software. TF2 cannot be set when either RCLK = 1 or CLK = 1

EXF2 T2CON. 6 Timer 2 external flag set when either a capture or reload is caused by a negative transition on T2EX, and EXEN2 = 1. When Timer 2 interrupt is enabled, EXF2 = 1 will cause the CPU to vector to the Timer 2 interrupt routine. EXF2 must be cleared by software.

RCLK T2CON. 5 Receive clock flag. When set, causes the Serial Port to use Timer 2 overflow pulses for its receive clock in modes 1 & 3. RCLK = 0 causes Timer 1 overflow to be used for the receive clock.

TLCK T2CON. 4 Transmit clock flag. When set, causes the Serial Port to use Timer 2 overflow pulses for its transmit clock in modes 1 & 3. TCLK = 0 causes Timer 1 overflows to be used for the transmit clock.

EXEN2 T2CON. 3 Timer 2 external enable flag. When set, allows a capture or reload to occur as a result of negative transition on T2EX if Timer 2 is not being used to clock the Serial Port. EXEN2 = 0 causes Timer 2 to ignore events at T2EX.

TR2 T2CON. 2 Software START/STOP control for Timer 2. A logic 1 starts the Timer.

C/$\overline{T2}$ T2CON. 1 Timer or Counter select.

0 = Internal Timer. 1 = External Event Counter (falling edge triggered).

CP/$\overline{RL2}$ T2CON. 0 Capture/Reload flag. When set, captures will occur on negative transitions at T2EX if EXEN2 = 1. When cleared, Auto-Reloads will occur either with Timer 2 overflows or negative transitions at T2EX when EXEN2 = 1. When either RCLK = 1 or TCLK = 1, this bit is ignored and the Timer is forced to Auto-Reload on Timer 2 overflow.

TIMER/COUNTER 2 SET-UP

Except for the baud rate generator mode, the values given for T2CON do not include the setting of the TR2 bit. Therefore, bit TR2 must be set, separately, to turn the Timer on.

As a Timer:

Table 7

MODE	T2CON	
	INTERNAL CONTROL (NOTE 1)	EXTERNAL CONTROL (NOTE 2)
16-bit Auto-Reload	00H	08H
16-bit Capture	01H	09H
BAUD rate generator receive & transmit same baud rate	34H	36H
receive only	24H	26H
transmit only	14H	16H

As a Counter:

Table 8

MODE	TMOD	
	INTERNAL CONTROL (NOTE 1)	EXTERNAL CONTROL (NOTE 2)
16-bit Auto-Reload	02H	0AH
16-bit Capture	03H	0BH

NOTES:
1. Capture/Reload occurs only on Timer/Counter overflow.
2. Capture/Reload occurs on Timer/Counter overflow and a 1 to 0 transition on T2EX (P1.1) pin except when Timer 2 is used in the baud rate generating mode.

SCON: SERIAL PORT CONTROL REGISTER. BIT ADDRESSABLE.

SM0	SM1	SM2	REN	TB8	RB8	TI	RI

SM0 SCON. 7 Serial Port mode specifier. (NOTE 1).

SM1 SCON. 6 Serial Port mode specifier. (NOTE 1).

SM2 SCON. 5 Enables the multiprocessor communication feature in modes 2 & 3. In mode 2 or 3, if SM2 is set to 1 then RI will not be activated if the received 9th data bit (RB8) is 0. In mode 1, if SM2 = 1 then RI will not be activated if a valid stop bit was not received. In mode 0, SM2 should be 0. (See Table 9).

REN SCON. 4 Set/Cleared by software to Enable/Disable reception.

TB8 SCON. 3 The 9th bit that will be transmitted in modes 2 & 3. Set/Cleared by software.

RB8 SCON. 2 In modes 2 & 3, is the 9th data bit that was received. In mode 1, if SM2 = 0, RB8 is the stop bit that was received. In mode 0, RB8 is not used.

TI SCON. 1 Transmit interrupt flag. Set by hardware at the end of the 8th bit time in mode 0, or at the beginning of the stop bit in the other modes. Must be cleared by software.

RI SCON. 0 Receive interrupt flag. Set by hardware at the end of the 8th bit time in mode 0, or halfway through the stop bit time in the other modes (except see SM2). Must be cleared by software.

NOTE 1:

SM0	SM1	Mode	Description	Baud Rate
0	0	0	SHIFT REGISTER	Fosc./12
0	1	1	8-Bit UART	Variable
1	0	2	9-Bit UART	Fosc./64 OR Fosc./32
1	1	3	9-Bit UART	Variable

SERIAL PORT SET-UP:

Table 9

MODE	SCON	SM2 VARIATION
0	10H	Single Processor Environment (SM2 = 0)
1	50H	
2	90H	
3	D0H	
0	NA	Multiprocessor Environment (SM2 = 1)
1	70H	
2	B0H	
3	F0H	

GENERATING BAUD RATES

Serial Port in Mode 0:

Mode 0 has a fixed baud rate which is 1/12 of the oscillator frequency. To run the serial port in this mode none of the Timer/Counters need to be set up. Only the SCON register needs to be defined.

$$\text{Baud Rate} = \frac{\text{Osc Freq}}{12}$$

Serial Port in Mode 1:

Mode 1 has a variable baud rate. The baud rate can be generated by either Timer 1 or Timer 2 (8052 only).

USING TIMER/COUNTER 1 TO GENERATE BAUD RATES:

For this purpose, Timer 1 is used in mode 2 (Auto-Reload). Refer to Timer Setup section of this chapter.

$$\text{Baud Rate} = \frac{\text{K x Oscillator Freq.}}{32 \times 12 \times [256 - (\text{TH1})]}$$

If SMOD = 0, then K = 1.
If SMOD = 1, then K = 2. (SMOD is the PCON register).

Most of the time the user knows the baud rate and needs to know the reload value for TH1. Therefore, the equation to calculate TH1 can be written as:

$$\text{TH1} = 256 - \frac{\text{K x Osc Freq.}}{384 \times \text{baud rate}}$$

TH1 must be an integer value. Rounding off TH1 to the nearest integer may not produce the desired baud rate. In this case, the user may have to choose another crystal frequency.

Since the PCON register is not bit addressable, one way to set the bit is logical ORing the PCON register. (ie, ORL PCON,#80H). The address of PCON is 87H.

USING TIMER/COUNTER 2 TO GENERATE BAUD RATES:

For this purpose, Timer 2 must be used in the baud rate generating mode. Refer to Timer 2 Setup Table in this chapter. If Timer 2 is being clocked through pin T2 (P1.0) the baud rate is:

$$\text{Baud Rate} = \frac{\text{Timer 2 Overflow Rate}}{16}$$

And if it is being clocked internally the baud rate is:

$$\text{Baud Rate} = \frac{\text{Osc Freq}}{32 \times [65536 - (\text{RCAP2H, RCAP2L})]}$$

To obtain the reload value for RCAP2H and RCAP2L the above equation can be rewritten as:

$$\text{RCAP2H, RCAP2L} = 65536 - \frac{\text{Osc Freq}}{32 \times \text{Baud Rate}}$$

SERIAL PORT IN MODE 2:

The baud rate is fixed in this mode and is $1/32$ or $1/64$ of the oscillator frequency depending on the value of the SMOD bit in the PCON register.

In this mode none of the Timers are used and the clock comes from the internal phase 2 clock.

SMOD = 1, Baud Rate = $1/32$ Osc Freq.

SMOD = 0, Baud Rate = $1/64$ Osc Freq.

To set the SMOD bit: ORL PCON,#80H. The address of PCON is 87H.

SERIAL PORT IN MODE 3:

The baud rate in mode 3 is variable and sets up exactly the same as in mode 1.

MCS®-51 INSTRUCTION SET

Table 10. 8051 Instruction Set Summary

Interrupt Response Time: Refer to Hardware Description Chapter.

Instructions that Affect Flag Settings(1)

Instruction	Flag			Instruction	Flag		
	C	OV	AC		C	OV	AC
ADD	X	X	X	CLR C	O		
ADDC	X	X	X	CPL C	X		
SUBB	X	X	X	ANL C,bit	X		
MUL	O	X		ANL C,/bit	X		
DIV	O	X		ORL C,bit	X		
DA	X			ORL C,bit	X		
RRC	X			MOV C,bit	X		
RLC	X			CJNE	X		
SETB C	1						

(1)Note that operations on SFR byte address 208 or bit addresses 209-215 (i.e., the PSW or bits in the PSW) will also affect flag settings.

Note on instruction set and addressing modes:

Rn — Register R7–R0 of the currently selected Register Bank.

direct — 8-bit internal data location's address. This could be an Internal Data RAM location (0–127) or a SFR [i.e., I/O port, control register, status register, etc. (128–255)].

@Ri — 8-bit internal data RAM location (0–255) addressed indirectly through register R1 or R0.

#data — 8-bit constant included in instruction.

#data 16 — 16-bit constant included in instruction.

addr 16 — 16-bit destination address. Used by LCALL & LJMP. A branch can be anywhere within the 64K-byte Program Memory address space.

addr 11 — 11-bit destination address. Used by ACALL & AJMP. The branch will be within the same 2K-byte page of program memory as the first byte of the following instruction.

rel — Signed (two's complement) 8-bit offset byte. Used by SJMP and all conditional jumps. Range is −128 to +127 bytes relative to first byte of the following instruction.

bit — Direct Addressed bit in Internal Data RAM or Special Function Register.

* — New operation not provided by 8048AH/8049AH.

Mnemonic		Description	Byte	Oscillator Period
ARITHMETIC OPERATIONS				
ADD	A,Rn	Add register to Accumulator	1	12
ADD	A,direct	Add direct byte to Accumulator	2	12
ADD	A,@Ri	Add indirect RAM to Accumulator	1	12
ADD	A,#data	Add immediate data to Accumulator	2	12
ADDC	A,Rn	Add register to Accumulator with Carry	1	12
ADDC	A,direct	Add direct byte to Accumulator with Carry	2	12
ADDC	A,@Ri	Add indirect RAM to Accumulator with Carry	1	12
ADDC	A,#data	Add immediate data to Acc with Carry	2	12
SUBB	A,Rn	Subtract Register from Acc with borrow	1	12
SUBB	A,direct	Subtract direct byte from Acc with borrow	2	12
SUBB	A,@Ri	Subtract indirect RAM from ACC with borrow	1	12
SUBB	A,#data	Subtract immediate data from Acc with borrow	2	12
INC	A	Increment Accumulator	1	12
INC	Rn	Increment register	1	12
INC	direct	Increment direct byte	2	12
INC	@Ri	Increment direct RAM	1	12
DEC	A	Decrement Accumulator	1	12
DEC	Rn	Decrement Register	1	12
DEC	direct	Decrement direct byte	2	12
DEC	@Ri	Decrement indirect RAM	1	12

All mnemonics copyrighted ©Intel Corporation 1980

Table 10. 8051 Instruction Set Summary (Continued)

Mnemonic		Description	Byte	Oscillator Period
ARITHMETIC OPERATIONS (Continued)				
INC	DPTR	Increment Data Pointer	1	24
MUL	AB	Multiply A & B	1	48
DIV	AB	Divide A by B	1	48
DA	A	Decimal Adjust Accumulator	1	12
LOGICAL OPERATIONS				
ANL	A,Rn	AND Register to Accumulator	1	12
ANL	A,direct	AND direct byte to Accumulator	2	12
ANL	A,@Ri	AND indirect RAM to Accumulator	1	12
ANL	A,#data	AND immediate data to Accumulator	2	12
ANL	direct,A	AND Accumulator to direct byte	2	12
ANL	direct,#data	AND immediate data to direct byte	3	24
ORL	A,Rn	OR register to Accumulator	1	12
ORL	A,direct	OR direct byte to Accumulator	2	12
ORL	A,@Ri	OR indirect RAM to Accumulator	1	12
ORL	A,#data	OR immediate data to Accumulator	2	12
ORL	direct,A	OR Accumulator to direct byte	2	12
ORL	direct,#data	OR immediate data to direct byte	3	24
XRL	A,Rn	Exclusive-OR register to Accumulator	1	12
XRL	A,direct	Exclusive-OR direct byte to Accumulator	2	12
XRL	A,@Ri	Exclusive-OR indirect RAM to Accumulator	1	12
XRL	A,#data	Exclusive-OR immediate data to Accumulator	2	12
XRL	direct,A	Exclusive-OR Accumulator to direct byte	2	12
XRL	direct,#data	Exclusive-OR immediate data to direct byte	3	24
CLR	A	Clear Accumulator	1	12
CPL	A	Complement Accumulator	1	12

Mnemonic		Description	Byte	Oscillator Period
LOGICAL OPERATIONS (Continued)				
RL	A	Rotate Accumulator Left	1	12
RLC	A	Rotate Accumulator Left through the Carry	1	12
RR	A	Rotate Accumulator Right	1	12
RRC	A	Rotate Accumulator Right through the Carry	1	12
SWAP	A	Swap nibbles within the Accumulator	1	12
DATA TRANSFER				
MOV	A,Rn	Move register to Accumulator	1	12
MOV	A,direct	Move direct byte to Accumulator	2	12
MOV	A,@Ri	Move indirect RAM to Accumulator	1	12
MOV	A,#data	Move immediate data to Accumulator	2	12
MOV	Rn,A	Move Accumulator to register	1	12
MOV	Rn,direct	Move direct byte to register	2	24
MOV	Rn,#data	Move immediate data to register	2	12
MOV	direct,A	Move Accumulator to direct byte	2	12
MOV	direct,Rn	Move register to direct byte	2	24
MOV	direct,direct	Move direct byte to direct	3	24
MOV	direct,@Ri	Move indirect RAM to direct byte	2	24
MOV	direct,#data	Move immediate data to direct byte	3	24
MOV	@Ri,A	Move Accumulator to indirect RAM	1	12

All mnemonics copyrighted ©Intel Corporation 1980

Table 10. 8051 Instruction Set Summary (Continued)

Mnemonic		Description	Byte	Oscillator Period
DATA TRANSFER (Continued)				
MOV	@Ri,direct	Move direct byte to indirect RAM	2	24
MOV	@Ri,#data	Move immediate data to indirect RAM	2	12
MOV	DPTR,#data16	Load Data Pointer with a 16-bit constant	3	24
MOVC	A,@A+DPTR	Move Code byte relative to DPTR to Acc	1	24
MOVC	A,@A+PC	Move Code byte relative to PC to Acc	1	24
MOVX	A,@Ri	Move External RAM (8-bit addr) to Acc	1	24
MOVX	A,@DPTR	Move External RAM (16-bit addr) to Acc	1	24
MOVX	@Ri,A	Move Acc to External RAM (8-bit addr)	1	24
MOVX	@DPTR,A	Move Acc to External RAM (16-bit addr)	1	24
PUSH	direct	Push direct byte onto stack	2	24
POP	direct	Pop direct byte from stack	2	24
XCH	A,Rn	Exchange register with Accumulator	1	12
XCH	A,direct	Exchange direct byte with Accumulator	2	12
XCH	A,@Ri	Exchange indirect RAM with Accumulator	1	12
XCHD	A,@Ri	Exchange low-order Digit indirect RAM with Acc	1	12

Mnemonic		Description	Byte	Oscillator Period
BOOLEAN VARIABLE MANIPULATION				
CLR	C	Clear Carry	1	12
CLR	bit	Clear direct bit	2	12
SETB	C	Set Carry	1	12
SETB	bit	Set direct bit	2	12
CPL	C	Complement Carry	1	12
CPL	bit	Complement direct bit	2	12
ANL	C,bit	AND direct bit to CARRY	2	24
ANL	C,/bit	AND complement of direct bit to Carry	2	24
ORL	C,bit	OR direct bit to Carry	2	24
ORL	C,/bit	OR complement of direct bit to Carry	2	24
MOV	C,bit	Move diret bit to Carry	2	12
MOV	bit,C	Move Carry to direct bit	2	24
JC	rel	Jump if Carry is set	2	24
JNC	rel	Jump if Carry not set	2	24
JB	bit,rel	Jump if direct Bit is set	3	24
JNB	bit,rel	Jump if direct Bit is Not set	3	24
JBC	bit,rel	Jump if direct Bit is set & clear bit	3	24
PROGRAM BRANCHING				
ACALL	addr11	Absolute Subroutine Call	2	24
LCALL	addr16	Long Subroutine Call	3	24
RET		Return from Subroutine	1	24
RETI		Return from interrupt	1	24
AJMP	addr11	Absolute Jump	2	24
LJMP	addr16	Long Jump	3	24
SJMP	rel	Short Jump (relative addr)	2	24

All mnemonics copyrighted ©Intel Corporation 1980

Table 10. 8051 Instruction Set Summary (Continued)

Mnemonic		Description	Byte	Oscillator Period
PROGRAM BRANCHING (Continued)				
JMP	@A+DPTR	Jump indirect relative to the DPTR	1	24
JZ	rel	Jump if Accumulator is Zero	2	24
JNZ	rel	Jump if Accumulator is Not Zero	2	24
CJNE	A,direct,rel	Compare direct byte to Acc and Jump if Not Equal	3	24
CJNE	A,#data,rel	Compare immediate to Acc and Jump if Not Equal	3	24
CJNE	Rn,#data,rel	Compare immediate to register and Jump if Not Equal	3	24
CJNE	@Ri,#data,rel	Compare immediate to indirect and Jump if Not Equal	3	24
DJNZ	Rn,rel	Decrement register and Jump if Not Zero	2	24
DJNZ	direct,rel	Decrement direct byte and Jump if Not Zero	3	24
NOP		No Operation	1	12

All mnemonics copyrighted ©Intel Corporation 1980

INSTRUCTION DEFINITIONS

ACALL addr11

Function: Absolute Call

Description: ACALL unconditionally calls a subroutine located at the indicated address. The instruction increments the PC twice to obtain the address of the following instruction, then pushes the 16-bit result onto the stack (low-order byte first) and increments the Stack Pointer twice. The destination address is obtained by successively concatenating the five high-order bits of the incremented PC, opcode bits 7-5, and the second byte of the instruction. The subroutine called must therefore start within the same 2K block of the program memory as the first byte of the instruction following ACALL. No flags are affected.

Example: Initially SP equals 07H. The label "SUBRTN" is at program memory location 0345 H. After executing the instruction,

ACALL SUBRTN

at location 0123H, SP will contain 09H, internal RAM locations 08H and 09H will contain 25H and 01H, respectively, and the PC will contain 0345H.

Bytes: 2

Cycles: 2

Encoding:

a10 a9 a8 1	0 0 0 1

a7 a6 a5 a4	a3 a2 a1 a0

Operation:
ACALL
$(PC) \leftarrow (PC) + 2$
$(SP) \leftarrow (SP) + 1$
$((SP)) \leftarrow (PC_{7\text{-}0})$
$(SP) \leftarrow (SP) + 1$
$((SP)) \leftarrow (PC_{15\text{-}8})$
$(PC_{10\text{-}0}) \leftarrow$ page address

ADD A,<src-byte>

Function: Add

Description: ADD adds the byte variable indicated to the Accumulator, leaving the result in the Accumulator. The carry and auxiliary-carry flags are set, respectively, if there is a carry-out from bit 7 or bit 3, and cleared otherwise. When adding unsigned integers, the carry flag indicates an overflow occured.

OV is set if there is a carry-out of bit 6 but not out of bit 7, or a carry-out of bit 7 but not bit 6; otherwise OV is cleared. When adding signed integers, OV indicates a negative number produced as the sum of two positive operands, or a positive sum from two negative operands.

Four source operand addressing modes are allowed: register, direct, register-indirect, or immediate.

Example: The Accumulator holds 0C3H (11000011B) and register 0 holds 0AAH (10101010B). The instruction,

ADD A,R0

will leave 6DH (01101101B) in the Accumulator with the AC flag cleared and both the carry flag and OV set to 1.

ADD A,Rn

Bytes: 1

Cycles: 1

Encoding:

0 0 1 0	1 r r r

Operation: ADD
(A) ← (A) + (Rn)

ADD A,direct

Bytes: 2

Cycles: 1

Encoding:

0 0 1 0	0 1 0 1	direct address

Operation: ADD
(A) ← (A) + (direct)

ADD A,@Ri

Bytes: 1

Cycles: 1

Encoding:

0 0 1 0	0 1 1 i

Operation: ADD
(A) ← (A) + ((Ri))

ADD A,#data

Bytes: 2

Cycles: 1

Encoding:

0 0 1 0	0 1 0 0	immediate data

Operation: ADD
(A) ← (A) + #data

ADDC A,<src-byte>

Function: Add with Carry

Description: ADDC simultaneously adds the byte variable indicated, the carry flag and the Accumulator contents, leaving the result in the Accumulator. The carry and auxiliary-carry flags are set, respectively, if there is a carry-out from bit 7 or bit 3, and cleared otherwise. When adding unsigned integers, the carry flag indicates an overflow occured.

OV is set if there is a carry-out of bit 6 but not out of bit 7, or a carry-out of bit 7 but not out of bit 6; otherwise OV is cleared. When adding signed integers, OV indicates a negative number produced as the sum of two positive operands or a positive sum from two negative operands.

Four source operand addressing modes are allowed: register, direct, register-indirect, or immediate.

Example: The Accumulator holds 0C3H (11000011B) and register 0 holds 0AAH (10101010B) with the carry flag set. The instruction,

ADDC A,R0

will leave 6EH (01101110B) in the Accumulator with AC cleared and both the Carry flag and OV set to 1.

ADDC A,Rn

Bytes: 1

Cycles: 1

Encoding:

0 0 1 1	1 r r r

Operation: ADDC
(A) ← (A) + (C) + (R_n)

ADDC A,direct

Bytes: 2

Cycles: 1

Encoding:

0 0 1 1	0 1 0 1	direct address

Operation: ADDC
(A) ← (A) + (C) + (direct)

ADDC A,@Ri

Bytes: 1

Cycles: 1

Encoding:

0 0 1 1	0 1 1 i

Operation: ADDC
(A) ← (A) + (C) + ((R_i))

ADDC A,#data

Bytes: 2

Cycles: 1

Encoding:

0 0 1 1	0 1 0 0	immediate data

Operation: ADDC
(A) ← (A) + (C) + #data

AJMP addr11

Function: Absolute Jump

Description: AJMP transfers program execution to the indicated address, which is formed at run-time by concatenating the high-order five bits of the PC (*after* incrementing the PC twice), opcode bits 7-5, and the second byte of the instruction. The destination must therefore be within the same 2K block of program memory as the first byte of the instruction following AJMP.

Example: The label "JMPADR" is at program memory location 0123H. The instruction,

AJMP JMPADR

is at location 0345H and will load the PC with 0123H.

Bytes: 2

Cycles: 2

Encoding:

a10 a9 a8 0	0 0 0 1	a7 a6 a5 a4	a3 a2 a1 a0

Operation: AJMP
(PC) ← (PC) + 2
($PC_{10\text{-}0}$) ← page address

ANL <dest-byte>,<src-byte>

Function: Logical-AND for byte variables

Description: ANL performs the bitwise logical-AND operation between the variables indicated and stores the results in the destination variable. No flags are affected.

The two operands allow six addressing mode combinations. When the destination is the Accumulator, the source can use register, direct, register-indirect, or immediate addressing; when the destination is a direct address, the source can be the Accumulator or immediate data.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, *not* the input pins.

Example: If the Accumulator holds 0C3H (11000011B) and register 0 holds 55H (01010101B) then the instruction,

ANL A,R0

will leave 41H (01000001B) in the Accumulator.

When the destination is a directly addressed byte, this instruction will clear combinations of bits in any RAM location or hardware register. The mask byte determining the pattern of bits to be cleared would either be a constant contained in the instruction or a value computed in the Accumulator at run-time. The instruction,

ANL P1,#01110011B

will clear bits 7, 3, and 2 of output port 1.

ANL A,Rn

Bytes: 1

Cycles: 1

Encoding:

0 1 0 1	1 r r r

Operation: ANL
(A) ← (A) ∧ (Rn)

ANL A,direct

Bytes: 2

Cycles: 1

Encoding:

0 1 0 1	0 1 0 1	direct address

Operation: ANL

(A) ← (A) ∧ (direct)

ANL A,@Ri

Bytes: 1

Cycles: 1

Encoding:

0 1 0 1	0 1 1 i

Operation: ANL

(A) ← (A) ∧ ((Ri))

ANL A,#data

Bytes: 2

Cycles: 1

Encoding:

0 1 0 1	0 1 0 0	immediate data

Operation: ANL
(A) ← (A) ∧ #data

ANL direct,A

Bytes: 2

Cycles: 1

Encoding:

0 1 0 1	0 0 1 0	direct address

Operation: ANL
(direct) ← (direct) ∧ (A)

ANL direct,#data

Bytes: 3

Cycles: 2

Encoding:

0 1 0 1	0 0 1 1	direct address	immediate data

Operation: ANL
(direct) ← (direct) ∧ #data

ANL C,<src-bit>

Function: Logical-AND for bit variables

Description: If the Boolean value of the source bit is a logical 0 then clear the carry flag; otherwise leave the carry flag in its current state. A slash ("/") preceding the operand in the assembly language indicates that the logical complement of the addressed bit is used as the source value, *but the source bit itself is not affected.* No other flags are affected.

Only direct addressing is allowed for the source operand.

Example: Set the carry flag if, and only if, P1.0 = 1, ACC. 7 = 1, and OV = 0:

```
MOV  C,P1.0     ;LOAD CARRY WITH INPUT PIN STATE
ANL  C,ACC.7    ;AND CARRY WITH ACCUM. BIT 7
ANL  C,/OV      ;AND WITH INVERSE OF OVERFLOW FLAG
```

ANL C,bit

Bytes: 2

Cycles: 2

Encoding:

1 0 0 0	0 0 1 0	bit address

Operation: ANL
(C) ← (C) ∧ (bit)

ANL C,/bit

Bytes: 2

Cycles: 2

Encoding:

1 0 1 1	0 0 0 0	bit address

Operation: ANL
(C) ← (C) ∧ ¬(bit)

CJNE <dest-byte>,<src-byte>, rel

Function: Compare and Jump if Not Equal.

Description: CJNE compares the magnitudes of the first two operands, and branches if their values are not equal. The branch destination is computed by adding the signed relative-displacement in the last instruction byte to the PC, after incrementing the PC to the start of the next instruction. The carry flag is set if the unsigned integer value of <dest-byte> is less than the unsigned integer value of <src-byte>; otherwise, the carry is cleared. Neither operand is affected.

The first two operands allow four addressing mode combinations: the Accumulator may be compared with any directly addressed byte or immediate data, and any indirect RAM location or working register can be compared with an immediate constant.

Example: The Accumulator contains 34H. Register 7 contains 56H. The first instruction in the sequence,

```
                CJNE    R7,#60H, NOT_EQ
;               ...     .....           ;  R7 = 60H.
NOT_EQ:         JC      REQ_LOW         ;  IF R7 < 60H.
;               ...     .....           ;  R7 > 60H.
```

sets the carry flag and branches to the instruction at label NOT_EQ. By testing the carry flag, this instruction determines whether R7 is greater or less than 60H.

If the data being presented to Port 1 is also 34H, then the instruction,

```
WAIT:  CJNE  A,P1,WAIT
```

clears the carry flag and continues with the next instruction in sequence, since the Accumulator does equal the data read from P1. (If some other value was being input on P1, the program will loop at this point until the P1 data changes to 34H.)

CJNE A,direct,rel

Bytes: 3

Cycles: 2

Encoding:

1 0 1 1	0 1 0 1	direct address	rel. address

Operation:

```
(PC) ← (PC) + 3
IF (A) <> (direct)
THEN
        (PC) ← (PC) + relative offset
IF (A) < (direct)
THEN
        (C) ← 1
ELSE
        (C) ← 0
```

CJNE A,#data,rel

Bytes: 3

Cycles: 2

Encoding:

1 0 1 1	0 1 0 0	immediate data	rel. address

Operation:

(PC) ← (PC) + 3
IF (A) <> *data*
THEN
(PC) ← (PC) + *relative offset*

IF (A) < *data*
THEN
(C) ← 1
ELSE
(C) ← 0

CJNE Rn,#data,rel

Bytes: 3

Cycles: 2

Encoding:

1 0 1 1	1 r r r	immediate data	rel. address

Operation:

(PC) ← (PC) + 3
IF (Rn) <> *data*
THEN
(PC) ← (PC) + *relative offset*

IF (Rn) < *data*
THEN
(C) ← 1
ELSE
(C) ← 0

CJNE @Ri,#data,rel

Bytes: 3

Cycles: 2

Encoding:

1 0 1 1	0 1 1 i	immediate data	rel. address

Operation:

(PC) ← (PC) + 3
IF ((Ri)) <> *data*
THEN
(PC) ← (PC) + *relative offset*

IF ((Ri)) < *data*
THEN
(C) ← 1
ELSE
(C) ← 0

CLR A

Function: Clear Accumulator

Description: The Accumulator is cleared (all bits set on zero). No flags are affected.

Example: The Accumulator contains 5CH (01011100B). The instruction,

```
CLR  A
```

will leave the Accumulator set to 00H (00000000B).

Bytes: 1

Cycles: 1

Encoding:

1 1 1 1	0 1 0 0

Operation:

```
CLR
(A) ← 0
```

CLR bit

Function: Clear bit

Description: The indicated bit is cleared (reset to zero). No other flags are affected. CLR can operate on the carry flag or any directly addressable bit.

Example: Port 1 has previously been written with 5DH (01011101B). The instruction,

```
CLR  P1.2
```

will leave the port set to 59H (01011001B).

CLR C

Bytes: 1

Cycles: 1

Encoding:

1 1 0 0	0 0 1 1

Operation:

```
CLR
(C) ← 0
```

CLR bit

Bytes: 2

Cycles: 1

Encoding:

1 1 0 0	0 0 1 0	bit address

Operation:

```
CLR
(bit) ← 0
```

CPL A

Function: Complement Accumulator

Description: Each bit of the Accumulator is logically complemented (one's complement). Bits which previously contained a one are changed to a zero and vice-versa. No flags are affected.

Example: The Accumulator contains 5CH (01011100B). The instruction,

```
CPL   A
```

will leave the Accumulator set to 0A3H (10100011B).

Bytes: 1

Cycles: 1

Encoding:

1 1 1 1	0 1 0 0

Operation:

```
CPL
(A) ← ¬(A)
```

CPL bit

Function: Complement bit

Description: The bit variable specified is complemented. A bit which had been a one is changed to zero and vice-versa. No other flags are affected. CLR can operate on the carry or any directly addressable bit.

Note: When this instruction is used to modify an output pin, the value used as the original data will be read from the output data latch, *not* the input pin.

Example: Port 1 has previously been written with 5BH (01011101B). The instruction sequence,

```
CPL   P1.1
CPL   P1.2
```

will leave the port set to 5BH (01011011B).

CPL C

Bytes: 1

Cycles: 1

Encoding:

1 0 1 1	0 0 1 1

Operation:

```
CPL
(C) ← ¬(C)
```

CPL bit

Bytes: 2

Cycles: 1

Encoding:

1 0 1 1	0 0 1 0	bit address

Operation: CPL
(bit) ← ¬ (bit)

DA A

Function: Decimal-adjust Accumulator for Addition

Description: DA A adjusts the eight-bit value in the Accumulator resulting from the earlier addition of two variables (each in packed-BCD format), producing two four-bit digits. Any ADD or ADDC instruction may have been used to perform the addition.

If Accumulator bits 3-0 are greater than nine (xxxx1010-xxxx1111), or if the AC flag is one, six is added to the Accumulator producing the proper BCD digit in the low-order nibble. This internal addition would set the carry flag if a carry-out of the low-order four-bit field propagated through all high-order bits, but it would not clear the carry flag otherwise.

If the carry flag is now set, or if the four high-order bits now exceed nine (1010xxxx-111xxxx), these high-order bits are incremented by six, producing the proper BCD digit in the high-order nibble. Again, this would set the carry flag if there was a carry-out of the high-order bits, but wouldn't clear the carry. The carry flag thus indicates if the sum of the original two BCD variables is greater than 100, allowing multiple precision decimal addition. OV is not affected.

All of this occurs during the one instruction cycle. Essentially, this instruction performs the decimal conversion by adding 00H, 06H, 60H, or 66H to the Accumulator, depending on initial Accumulator and PSW conditions.

Note: DA A *cannot* simply convert a hexadecimal number in the Accumulator to BCD notation, nor does DA A apply to decimal subtraction.

Example: The Accumulator holds the value 56H (01010110B) representing the packed BCD digits of the decimal number 56. Register 3 contains the value 67H (01100111B) representing the packed BCD digits of the decimal number 67. The carry flag is set. The instruction sequence.

```
ADDC  A,R3
DA    A
```

will first perform a standard twos-complement binary addition, resulting in the value 0BEH (10111110) in the Accumulator. The carry and auxiliary carry flags will be cleared.

The Decimal Adjust instruction will then alter the Accumulator to the value 24H (00100100B), indicating the packed BCD digits of the decimal number 24, the low-order two digits of the decimal sum of 56, 67, and the carry-in. The carry flag will be set by the Decimal Adjust instruction, indicating that a decimal overflow occurred. The true sum 56, 67, and 1 is 124.

BCD variables can be incremented or decremented by adding 01H or 99H. If the Accumulator initially holds 30H (representing the digits of 30 decimal), then the instruction sequence,

```
ADD   A,#99H
DA    A
```

will leave the carry set and 29H in the Accumulator, since 30 + 99 = 129. The low-order byte of the sum can be interpreted to mean 30 − 1 = 29.

Bytes: 1

Cycles: 1

Encoding:

1 1 0 1	0 1 0 0

Operation: DA

-contents of Accumulator are BCD

IF $[[(A_{3-0}) > 9] \vee [(AC) = 1]]$

THEN $(A_{3-0}) \leftarrow (A_{3-0}) + 6$

AND

IF $[[(A_{7-4}) > 9] \vee [(C) = 1]]$

THEN $(A_{7-4}) \leftarrow (A_{7-4}) + 6$

DEC byte

Function: Decrement

Description: The variable indicated is decremented by 1. An original value of 00H will underflow to 0FFH. No flags are affected. Four operand addressing modes are allowed: accumulator, register, direct, or register-indirect.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, *not* the input pins.

Example: Register 0 contains 7FH (01111111B). Internal RAM locations 7EH and 7FH contain 00H and 40H, respectively. The instruction sequence,

```
DEC  @R0

DEC  R0

DEC  @R0
```

will leave register 0 set to 7EH and internal RAM locations 7EH and 7FH set to 0FFH and 3FH.

DEC A

Bytes: 1

Cycles: 1

Encoding:

0 0 0 1	0 1 0 0

Operation: DEC
(A) ← (A) − 1

DEC Rn

Bytes: 1

Cycles: 1

Encoding:

0 0 0 1	1 r r r

Operation: DEC
(Rn) ← (Rn) − 1

DEC direct

Bytes: 2

Cycles: 1

Encoding:

0 0 0 1	0 1 0 1	direct address

Operation: DEC
(direct) ← (direct) − 1

DEC @Ri

Bytes: 1

Cycles: 1

Encoding:

0 0 0 1	0 1 1 i

Operation: DEC
((Ri)) ← ((Ri)) − 1

DIV AB

Function: Divide

Description: DIV AB divides the unsigned eight-bit integer in the Accumulator by the unsigned eight-bit integer in register B. The Accumulator receives the integer part of the quotient; register B receives the integer remainder. The carry and OV flags will be cleared.

Exception: if B had originally contained 00H, the values returned in the Accumulator and B-register will be undefined and the overflow flag will be set. The carry flag is cleared in any case.

Example: The Accumulator contains 251 (0FBH or 11111011B) and B contains 18 (12H or 00010010B). The instruction,

DIV AB

will leave 13 in the Accumulator (0DH or 00001101B) and the value 17 (11H or 00010001B) in B, since 251 = (13 X 18) + 17. Carry and OV will both be cleared.

Bytes: 1

Cycles: 4

Encoding:

1 0 0 0	0 1 0 0

Operation: DIV
$(A)_{15\text{-}8}$
$(B)_{7\text{-}0}$ ← (A)/(B)

DJNZ <byte>,<rel-addr>

Function: Decrement and Jump if Not Zero

Description: DJNZ decrements the location indicated by 1, and branches to the address indicated by the second operand if the resulting value is not zero. An original value of 00H will underflow to 0FFH. No flags are affected. The branch destination would be computed by adding the signed relative-displacement value in the last instruction byte to the PC, after incrementing the PC to the first byte of the following instruction.

The location decremented may be a register or directly addressed byte.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, *not* the input pins.

Example: Internal RAM locations 40H, 50H, and 60H contain the values 01H, 70H, and 15H, respectively. The instruction sequence,

```
DJNZ  40H,LABEL_1
DJNZ  50H,LABEL_2
DJNZ  60H,LABEL_3
```

will cause a jump to the instruction at label LABEL_2 with the values 00H, 6FH, and 15H in the three RAM locations. The first jump was *not* taken because the result was zero.

This instruction provides a simple way of executing a program loop a given number of times, or for adding a moderate time delay (from 2 to 512 machine cycles) with a single instruction. The instruction sequence,

```
        MOV     R2,#8
TOGGLE: CPL     P1.7
        DJNZ    R2,TOGGLE
```

will toggle P1.7 eight times, causing four output pulses to appear at bit 7 of output Port 1. Each pulse will last three machine cycles; two for DJNZ and one to alter the pin.

DJNZ Rn,rel

Bytes: 2

Cycles: 2

Encoding:

1 1 0 1	1 r r r	rel. address

Operation:

```
DJNZ
(PC) ← (PC) + 2
(Rn) ← (Rn) − 1
IF  (Rn) > 0 or (Rn) < 0
    THEN
          (PC) ← (PC) + rel
```

DJNZ direct,rel

Bytes: 3

Cycles: 2

Encoding:

1 1 0 1	0 1 0 1	direct address	rel. address

Operation:

```
DJNZ
(PC) ← (PC) + 2
(direct) ← (direct) − 1
IF (direct) > 0 or (direct) < 0
   THEN
        (PC) ← (PC) + rel
```

INC <byte>

Function: Increment

Description: INC increments the indicated variable by 1. An original value of 0FFH will overflow to 00H. No flags are affected. Three addressing modes are allowed: register, direct, or register-indirect.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, *not* the input pins.

Example: Register 0 contains 7EH (011111110B). Internal RAM locations 7EH and 7FH contain 0FFH and 40H, respectively. The instruction sequence,

```
INC  @R0
INC  R0
INC  @R0
```

will leave register 0 set to 7FH and internal RAM locations 7EH and 7FH holding (respectively) 00H and 41H.

INC A

Bytes: 1

Cycles: 1

Encoding:

0 0 0 0	0 1 0 0

Operation:

```
INC
(A) ← (A) + 1
```

INC Rn

Bytes: 1

Cycles: 1

Encoding:

0 0 0 0	1 r r r

Operation: INC
(Rn) ← (Rn) + 1

INC direct

Bytes: 2

Cycles: 1

Encoding:

0 0 0 0	0 1 0 1	direct address

Operation: INC
(direct) ← (direct) + 1

INC @Ri

Bytes: 1

Cycles: 1

Encoding:

0 0 0 0	0 1 1 i

Operation: INC
((Ri)) ← ((Ri)) + 1

INC DPTR

Function: Increment Data Pointer

Description: Increment the 16-bit data pointer by 1. A 16-bit increment (modulo 2^{16}) is performed; an overflow of the low-order byte of the data pointer (DPL) from 0FFH to 00H will increment the high-order byte (DPH). No flags are affected.

This is the only 16-bit register which can be incremented.

Example: Registers DPH and DPL contain 12H and 0FEH, respectively. The instruction sequence,

```
INC   DPTR
INC   DPTR
INC   DPTR
```

will change DPH and DPL to 13H and 01H.

Bytes: 1

Cycles: 2

Encoding:

1 0 1 0	0 0 1 1

Operation: INC
(DPTR) ← (DPTR) + 1

JB bit,rel

Function: Jump if Bit set

Description: If the indicated bit is a one, jump to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. *The bit tested is not modified.* No flags are affected.

Example: The data present at input port 1 is 11001010B. The Accumulator holds 56 (01010110B). The instruction sequence,

```
JB   P1.2,LABEL1

JB   ACC.2,LABEL2
```

will cause program execution to branch to the instruction at label LABEL2.

Bytes: 3

Cycles: 2

Encoding:

0 0 1 0	0 0 0 0	bit address	rel. address

Operation:

```
JB
(PC) ← (PC) + 3
IF  (bit) = 1
    THEN
            (PC) ← (PC) + rel
```

JBC bit,rel

Function: Jump if Bit is set and Clear bit

Description: If the indicated bit is one, branch to the address indicated; otherwise proceed with the next instruction. *The bit will not be cleared if it is already a zero.* The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. No flags are affected.

Note: When this instruction is used to test an output pin, the value used as the original data will be read from the output data latch, *not* the input pin.

Example: The Accumulator holds 56H (01010110B). The instruction sequence,

```
JBC  ACC.3,LABEL1
JBC  ACC.2,LABEL2
```

will cause program execution to continue at the instruction identified by the label LABEL2, with the Accumulator modified to 52H (01010010B).

Bytes: 3

Cycles: 2

Encoding:

0 0 0 1	0 0 0 0	bit address	rel. address

Operation:

```
JBC
(PC) ← (PC) + 3
IF  (bit) = 1
    THEN
        (bit) ← 0
        (PC) ← (PC) + rel
```

JC rel

Function: Jump if Carry is set

Description: If the carry flag is set, branch to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice. No flags are affected.

Example: The carry flag is cleared. The instruction sequence,

```
JC    LABEL1
CPL   C
JC    LABEL 2
```

will set the carry and cause program execution to continue at the instruction identified by the label LABEL2.

Bytes: 2

Cycles: 2

Encoding:

0 1 0 0	0 0 0 0	rel. address

Operation:

```
JC
(PC) ← (PC) + 2
IF  (C) = 1
    THEN
        (PC) ← (PC) + rel
```

JMP @A+DPTR

Function: Jump indirect

Description: Add the eight-bit unsigned contents of the Accumulator with the sixteen-bit data pointer, and load the resulting sum to the program counter. This will be the address for subsequent instruction fetches. Sixteen-bit addition is performed (modulo 2^{16}): a carry-out from the low-order eight bits propagates through the higher-order bits. Neither the Accumulator nor the Data Pointer is altered. No flags are affected.

Example: An even number from 0 to 6 is in the Accumulator. The following sequence of instructions will branch to one of four AJMP instructions in a jump table starting at JMP_TBL:

```
        MOV     DPTR,#JMP_TBL
        JMP     @A+DPTR
JMP_TBL: AJMP   LABEL0
        AJMP    LABEL1
        AJMP    LABEL2
        AJMP    LABEL3
```

If the Accumulator equals 04H when starting this sequence, execution will jump to label LABEL2. Remember that AJMP is a two-byte instruction, so the jump instructions start at every other address.

Bytes: 1

Cycles: 2

Encoding:

0 1 1 1	0 0 1 1

Operation: JMP
(PC) ← (A) + (DPTR)

JNB bit,rel

Function: Jump if Bit Not set

Description: If the indicated bit is a zero, branch to the indicated address; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. *The bit tested is not modified.* No flags are affected.

Example: The data present at input port 1 is 11001010B. The Accumulator holds 56H (01010110B). The instruction sequence,

```
JNB   P1.3,LABEL1
JNB   ACC.3,LABEL2
```

will cause program execution to continue at the instruction at label LABEL2.

Bytes: 3

Cycles: 2

Encoding:

0 0 1 1	0 0 0 0	bit address	rel. address

Operation:

```
JNB
(PC) ← (PC) + 3
IF  (bit) = 0
    THEN (PC) ← (PC) + rel.
```

JNC rel

Function: Jump if Carry not set

Description: If the carry flag is a zero, branch to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice to point to the next instruction. The carry flag is not modified.

Example: The carry flag is set. The instruction sequence,

```
JNC   LABEL1
CPL   C
JNC   LABEL2
```

will clear the carry and cause program execution to continue at the instruction identified by the label LABEL2.

Bytes: 2

Cycles: 2

Encoding:

0 1 0 1	0 0 0 0	rel. address

Operation:

```
JNC
(PC) ← (PC) + 2
IF  (C) = 0
    THEN  (PC) ← (PC) + rel
```

JNZ rel

Function: Jump if Accumulator Not Zero

Description: If any bit of the Accumulator is a one, branch to the indicated address; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice. The Accumulator is not modified. No flags are affected.

Example: The Accumulator originally holds 00H. The instruction sequence,

```
JNZ  LABEL1
INC  A
JNZ  LABEL2
```

will set the Accumulator to 01H and continue at label LABEL2.

Bytes: 2

Cycles: 2

Encoding:

0 1 1 1	0 0 0 0	rel. address

Operation:

```
JNZ
(PC) ← (PC) + 2
IF    (A) ≠ 0
      THEN  (PC) ← (PC) + rel
```

JZ rel

Function: Jump if Accumulator Zero

Description: If all bits of the Accumulator are zero, branch to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice. The Accumulator is not modified. No flags are affected.

Example: The Accumulator originally contains 01H. The instruction sequence,

```
JZ   LABEL1
DEC  A
JZ   LABEL2
```

will change the Accumulator to 00H and cause program execution to continue at the instruction identified by the label LABEL2.

Bytes: 2

Cycles: 2

Encoding:

0 1 1 0	0 0 0 0	rel. address

Operation:

```
JZ
(PC) ← (PC) + 2
IF    (A) = 0
      THEN  (PC) ← (PC) + rel
```

LCALL addr16

Function: Long call

Description: LCALL calls a subroutine located at the indicated address. The instruction adds three to the program counter to generate the address of the next instruction and then pushes the 16-bit result onto the stack (low byte first), incrementing the Stack Pointer by two. The high-order and low-order bytes of the PC are then loaded, respectively, with the second and third bytes of the LCALL instruction. Program execution continues with the instruction at this address. The subroutine may therefore begin anywhere in the full 64K-byte program memory address space. No flags are affected.

Example: Initially the Stack Pointer equals 07H. The label "SUBRTN" is assigned to program memory location 1234H. After executing the instruction,

```
LCALL  SUBRTN
```

at location 0123H, the Stack Pointer will contain 09H, internal RAM locations 08H and 09H will contain 26H and 01H, and the PC will contain 1235H.

Bytes: 3

Cycles: 2

Encoding:

0 0 0 1	0 0 1 0	addr15-addr8	addr7-addr0

Operation:

```
LCALL
(PC) ← (PC) + 3
(SP) ← (SP) + 1
((SP)) ← (PC7-0)
(SP) ← (SP) + 1
((SP)) ← (PC15-8)
(PC) ← addr15-0
```

LJMP addr16

Function: Long Jump

Description: LJMP causes an unconditional branch to the indicated address, by loading the high-order and low-order bytes of the PC (respectively) with the second and third instruction bytes. The destination may therefore be anywhere in the full 64K program memory address space. No flags are affected.

Example: The label "JMPADR" is assigned to the instruction at program memory location 1234H. The instruction,

```
LJMP  JMPADR
```

at location 0123H will load the program counter with 1234H.

Bytes: 3

Cycles: 2

Encoding:

0 0 0 0	0 0 1 0	addr15-addr8	addr7-addr0

Operation:

```
LJMP
(PC) ← addr15-0
```

MOV <dest-byte>,<src-byte>

Function: Move byte variable

Description: The byte variable indicated by the second operand is copied into the location specified by the first operand. The source byte is not affected. No other register or flag is affected.

This is by far the most flexible operation. Fifteen combinations of source and destination addressing modes are allowed.

Example: Internal RAM location 30H holds 40H. The value of RAM location 40H is 10H. The data present at input port 1 is 11001010B (0CAH).

```
MOV   R0,#30H   ;R0 <= 30H
MOV   A,@R0     ;A <= 40H
MOV   R1,A      ;R1 <= 40H
MOV   R,@R1     ;B <= 10H
MOV   @R1,P1    ;RAM (40H) <= 0CAH
MOV   P2,P1     ;P2 #0CAH
```

leaves the value 30H in register 0, 40H in both the Accumulator and register 1, 10H in register B, and 0CAH (11001010B) both in RAM location 40H and output on port 2.

MOV A,Rn

Bytes: 1

Cycles: 1

Encoding:

1 1 1 0	1 r r r

Operation: MOV
(A) ← (Rn)

*MOV A,direct

Bytes: 2

Cycles: 1

Encoding:

1 1 1 0	0 1 0 1	direct address

Operation: MOV
(A) ← (direct)

MOV A,ACC is not a valid instruction.

MOV A,@Ri

Bytes: 1

Cycles: 1

Encoding:

1 1 1 0	0 1 1 i

Operation: MOV
(A) ← ((Ri))

MOV A,#data

Bytes: 2

Cycles: 1

Encoding:

0 1 1 1	0 1 0 0	immediate data

Operation: MOV
(A) ← #data

MOV Rn,A

Bytes: 1

Cycles: 1

Encoding:

1 1 1 1	1 r r r

Operation: MOV
(Rn) ← (A)

MOV Rn,direct

Bytes: 2

Cycles: 2

Encoding:

1 0 1 0	1 r r r	direct addr.

Operation: MOV
(Rn) ← (direct)

MOV Rn,#data

Bytes: 2

Cycles: 1

Encoding:

0 1 1 1	1 r r r	immediate data

Operation: MOV
(Rn) ← #data

MOV direct,A

Bytes: 2

Cycles: 1

Encoding:

1 1 1 1	0 1 0 1	direct address

Operation: MOV
(direct) ← (A)

MOV direct,Rn

Bytes: 2

Cycles: 2

Encoding:

1 0 0 0	1 r r r	direct address

Operation: MOV
(direct) ← (Rn)

MOV direct,direct

Bytes: 3

Cycles: 2

Encoding:

1 0 0 0	0 1 0 1	dir. addr. (src)	dir. addr. (dest)

Operation: MOV
(direct) ← (direct)

MOV direct,@Ri

Bytes: 2

Cycles: 2

Encoding:

1 0 0 0	0 1 1 i	direct addr.

Operation: MOV
(direct) ← ((Ri))

MOV direct,#data

Bytes: 3

Cycles: 2

Encoding:

0 1 1 1	0 1 0 1	direct address	immediate data

Operation: MOV
(direct) ← #data

MOV @Ri,A

Bytes: 1

Cycles: 1

Encoding:

1 1 1 1	0 1 1 i

Operation: MOV
((Ri)) ← (A)

MOV @Ri,direct

Bytes: 2

Cycles: 2

Encoding:

1 0 1 0	0 1 1 i	direct addr.

Operation: MOV
((Ri)) ← (direct)

MOV @Ri,#data

Bytes: 2

Cycles: 1

Encoding:

0 1 1 1	0 1 1 i	immediate data

Operation: MOV
((RI)) ← #data

MOV <dest-bit>,<src-bit>

Function: Move bit data

Description: The Boolean variable indicated by the second operand is copied into the location specified by the first operand. One of the operands must be the carry flag; the other may be any directly addressable bit. No other register or flag is affected.

Example: The carry flag is originally set. The data present at input Port 3 is 11000101B. The data previously written to output Port 1 is 35H (00110101B).

```
MOV  P1.3,C
MOV  C,P3.3
MOV  P1.2,C
```

will leave the carry cleared and change Port 1 to 39H (00111001B).

MOV C,bit

Bytes: 2

Cycles: 1

Encoding:

1 0 1 0	0 0 1 0	bit address

Operation: MOV
(C) ← (bit)

MOV bit,C

Bytes: 2

Cycles: 2

Encoding:

1 0 0 1	0 0 1 0	bit address

Operation: MOV
(bit) ← (C)

MOV DPTR,#data16

Function: Load Data Pointer with a 16-bit constant

Description: The Data Pointer is loaded with the 16-bit constant indicated. The 16-bit constant is loaded into the second and third bytes of the instruction. The second byte (DPH) is the high-order byte, while the third byte (DPL) holds the low-order byte. No flags are affected.

This is the only instruction which moves 16 bits of data at once.

Example: The instruction,

MOV DPTR,#1234H

will load the value 1234H into the Data Pointer: DPH will hold 12H and DPL will hold 34H.

Bytes: 3

Cycles: 2

Encoding:

1 0 0 1	0 0 0 0	immed. data15-8	immed. data7-0

Operation: MOV
(DPTR) ← #data$_{15\text{-}0}$
DPH ☐ DPL ← #data$_{15\text{-}8}$ ☐ #data$_{7\text{-}0}$

MOVC A,@A+<base-reg>

Function: Move Code byte

Description: The MOVC instructions load the Accumulator with a code byte, or constant from program memory. The address of the byte fetched is the sum of the original unsigned eight-bit Accumulator contents and the contents of a sixteen-bit base register, which may be either the Data Pointer or the PC. In the latter case, the PC is incremented to the address of the following instruction before being added with the Accumulator; otherwise the base register is not altered. Sixteen-bit addition is performed so a carry-out from the low-order eight bits may propagate through higher-order bits. No flags are affected.

Example: A value between 0 and 3 is in the Accumulator. The following instructions will translate the value in the Accumulator to one of four values defined by the DB (define byte) directive.

```
REL_PC: INC     A
        MOVC    A,@A+PC
        RET
        DB      66H
        DB      77H
        DB      88H
        DB      99H
```

If the subroutine is called with the Accumulator equal to 01H, it will return with 77H in the Accumulator. The INC A before the MOVC instruction is needed to "get around" the RET instruction above the table. If several bytes of code separated the MOVC from the table, the corresponding number would be added to the Accumulator instead.

MOVC A,@A+DPTR

Bytes: 1

Cycles: 2

Encoding:

1 0 0 1	0 0 1 1

Operation: MOVC
(A) ← ((A) + (DPTR))

MOVC A,@A + PC

Bytes: 1

Cycles: 2

Encoding:

1 0 0 0	0 0 1 1

Operation: MOVC
(PC) ← (PC) + 1
(A) ← ((A) + (PC))

MOVX <dest-byte>,<src-byte>

Function: Move External

Description: The MOVX instructions transfer data between the Accumulator and a byte of external data memory, hence the "X" appended to MOV. There are two types of instructions, differing in whether they provide an eight-bit or sixteen-bit indirect address to the external data RAM.

In the first type, the contents of R0 or R1 in the current register bank provide an eight-bit address multiplexed with data on P0. Eight bits are sufficient for external I/O expansion decoding or for a relatively small RAM array. For somewhat larger arrays, any output port pins can be used to output higher-order address bits. These pins would be controlled by an output instruction preceding the MOVX.

In the second type of MOVX instruction, the Data Pointer generates a sixteen-bit address. P2 outputs the high-order eight address bits (the contents of DPH) while P0 multiplexes the low-order eight bits (DPL) with data. The P2 Special Function Register retains its previous contents while the P2 output buffers are emitting the contents of DPH. This form is faster and more efficient when accessing very large data arrays (up to 64K bytes), since no additional instructions are needed to set up the output ports.

It is possible in some situations to mix the two MOVX types. A large RAM array with its high-order address lines driven by P2 can be addressed via the Data Pointer, or with code to output high-order address bits to P2 followed by a MOVX instruction using R0 or R1.

Example: An external 256 byte RAM using multiplexed address/data lines (e.g., an Intel 8155 RAM/I/O/Timer) is connected to the 8051 Port 0. Port 3 provides control lines for the external RAM. Ports 1 and 2 are used for normal I/O. Registers 0 and 1 contain 12H and 34H. Location 34H of the external RAM holds the value 56H. The instruction sequence,

```
MOVX  A,@R1
MOVX  @R0,A
```

copies the value 56H into both the Accumulator and external RAM location 12H.

MOVX A,@Ri

Bytes: 1

Cycles: 2

Encoding:

1 1 1 0	0 0 1 i

Operation: MOVX
(A) ← ((Ri))

MOVX A,@DPTR

Bytes: 1

Cycles: 2

Encoding:

1 1 1 0	0 0 0 0

Operation: MOVX
(A) ← ((DPTR))

MOVX @Ri,A

Bytes: 1

Cycles: 2

Encoding:

1 1 1 1	0 0 1 i

Operation: MOVX
((Ri)) ← (A)

MOVX @DPTR,A

Bytes: 1

Cycles: 2

Encoding:

1 1 1 1	0 0 0 0

Operation: MOVX
(DPTR) ← (A)

NOP

Function: No Operation

Description: Execution continues at the following instruction. Other than the PC, no registers or flags are affected.

Example: It is desired to produce a low-going output pulse on bit 7 of Port 2 lasting exactly 5 cycles. A simple SETB/CLR sequence would generate a one-cycle pulse, so four additional cycles must be inserted. This may be done (assuming no interrupts are enabled) with the instruction sequence,

```
CLR   P2.7
NOP
NOP
NOP
NOP
SETB  P2.7
```

Bytes: 1

Cycles: 1

Encoding:

0 0 0 0	0 0 0 0

Operation: NOP
(PC) ← (PC) + 1

MUL AB

Function: Multiply

Description: MUL AB multiplies the unsigned eight-bit integers in the Accumulator and register B. The low-order byte of the sixteen-bit product is left in the Accumulator, and the high-order byte in B. If the product is greater than 255 (0FFH) the overflow flag is set; otehrwise it is cleared. The carry flag is always cleared.

Example: Originally the Accumulator holds the value 80 (50H). Register B holds the value 160 (0A0H). The instruction,

```
MUL   AB
```

will give the product 12,800 (3200H), so B is changed to 32H (00110010B) and the Accumulator is cleared. The overflow flag is set, carry is cleared.

Bytes: 1

Cycles: 4

Encoding:

1 0 1 0	0 1 0 0

Operation: MUL
$(A)_{7-0}$ ← (A) X (B)
$(B)_{15-8}$

ORL <dest-byte> <src-byte>

Function: Logical-OR for byte variables

Description: ORL performs the bitwise logical-OR operation between the indicated variables, storing the results in the destination byte. No flags are affected.

The two operands allow six addressing mode combinations. When the destination is the Accumulator, the source can use register, direct, register-indirect, or immediate addressing; when the destination is a direct address, the source can be the Accumulator or immediate data.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, *not* the input pins.

Example: If the Accumulator holds 0C3H (11000011B) and R0 holds 55H (01010101B) then the instruction,

```
ORL   A,R0
```

will leave the Accumulator holding the value 0D7H (11010111B).

When the destination is a directly addressed byte, the instruction can set combinations of bits in any RAM location or hardware register. The pattern of bits to be set is determined by a mask byte, which may be either a constant data value in the instruction or a variable computed in the Accumulator at run-time. The instruction,

```
ORL   P1,#00110010B
```

will set bits 5, 4, and 1 of output Port 1.

ORL A,Rn

Bytes: 1

Cycles: 1

Encoding:

0 1 0 0	1 r r r

Operation: ORL
(A) ← (A) ∨ (Rn)

ORL A,direct

Bytes: 2

Cycles: 1

Encoding:

0 1 0 0	0 1 0 1	direct address

Operation: ORL
(A) ← (A) ∨ (direct)

ORL A,@Ri

Bytes: 1

Cycles: 1

Encoding:

0 1 0 0	0 1 1 i

Operation: ORL
(A) ← (A) ∨ ((Ri))

ORL A,#data

Bytes: 2

Cycles: 1

Encoding:

0 1 0 0	0 1 0 0	immediate data

Operation: ORL
(A) ← (A) ∨ #data

ORL direct,A

Bytes: 2

Cycles: 1

Encoding:

0 1 0 0	0 0 1 0	direct address

Operation: ORL
(direct) ← (direct) ∨ (A)

ORL direct,#data

Bytes: 3

Cycles: 2

Encoding:

0 1 0 0	0 0 1 1	direct addr.	immediate data

Operation: ORL
(direct) ← (direct) ∨ #data

ORL C,<src-bit>

Function: Logical-OR for bit variables

Description: Set the carry flag if the Boolean value is a logical 1; leave the carry in its current state otherwise . A slash ("/") preceding the operand in the assembly language indicates that the logical complement of the addressed bit is used as the source value, but the source bit itself is not affected. No other flags are affected.

Example: Set the carry flag if and only if P1.0 = 1, ACC. 7 = 1, or OV = 0:

```
MOV   C,P1.0     ;LOAD CARRY WITH INPUT PIN P10
ORL   C,ACC.7    ;OR CARRY WITH THE ACC. BIT 7
ORL   C,/OV      ;OR CARRY WITH THE INVERSE OF OV.
```

ORL C,bit

Bytes: 2

Cycles: 2

Encoding:

0 1 1 1	0 0 1 0	bit address

Operation: ORL
(C) ← (C) ∨ (bit)

ORL C,/bit

Bytes: 2

Cycles: 2

Encoding:

1 0 1 0	0 0 0 0	bit address

Operation: ORL
(C) ← (C) ∨ $\overline{(bit)}$

POP direct

Function: Pop from stack.

Description: The contents of the internal RAM location addressed by the Stack Pointer is read, and the Stack Pointer is decremented by one. The value read is then transferred to the directly addressed byte indicated. No flags are affected.

Example: The Stack Pointer originally contains the value 32H, and internal RAM locations 30H through 32H contain the values 20H, 23H, and 01H, respectively. The instruction sequence,

```
POP   DPH
POP   DPL
```

will leave the Stack Pointer equal to the value 30H and the Data Pointer set to 0123H. At this point the instruction,

```
POP   SP
```

will leave the Stack Pointer set to 20H. Note that in this special case the Stack Pointer was decremented to 2FH before being loaded with the value popped (20H).

Bytes: 2

Cycles: 2

Encoding:

1 1 0 1	0 0 0 0	direct address

Operation:

```
POP
(direct) ← ((SP))
(SP) ← (SP) − 1
```

PUSH direct

Function: Push onto stack

Description: The Stack Pointer is incremented by one. The contents of the indicated variable is then copied into the internal RAM location addressed by the Stack Pointer. Otherwise no flags are affected.

Example: On entering an interrupt routine the Stack Pointer contains 09H. The Data Pointer holds the value 0123H. The instruction sequence,

```
PUSH   DPL
PUSH   DPH
```

will leave the Stack Pointer set to 0BH and store 23H and 01H in internal RAM locations 0AH and 0BH, respectively.

Bytes: 2

Cycles: 2

Encoding:

1 1 0 0	0 0 0 0	direct address

Operation:

```
PUSH
(SP) ← (SP) + 1
((SP)) ← (direct)
```

RET

Function: Return from subroutine

Description: RET pops the high- and low-order bytes of the PC successively from the stack, decrementing the Stack Pointer by two. Program execution continues at the resulting address, generally the instruction immediately following an ACALL or LCALL. No flags are affected.

Example: The Stack Pointer originally contains the value 0BH. Internal RAM locations 0AH and 0BH contain the values 23H and 01H, respectively. The instruction,

RET

will leave the Stack Pointer equal to the value 09H. Program execution will continue at location 0123H.

Bytes: 1

Cycles: 2

Encoding:

0 0 1 0	0 0 1 0

Operation: RET
$(PC_{15-8}) \leftarrow ((SP))$
$(SP) \leftarrow (SP) - 1$
$(PC_{7-0}) \leftarrow ((SP))$
$(SP) \leftarrow (SP) - 1$

RETI

Function: Return from interrupt

Description: RETI pops the high- and low-order bytes of the PC successively from the stack, and restores the interrupt logic to accept additional interrupts at the same priority level as the one just processed. The Stack Pointer is left decremented by two. No other registers are affected; the PSW is *not* automatically restored to its pre-interrupt status. Program execution continues at the resulting address, which is generally the instruction immediately after the point at which the interrupt request was detected. If a lower- or same-level interrupt had been pending when the RETI instruction is executed, that one instruction will be executed before the pending interrupt is processed.

Example: The Stack Pointer originally contains the value 0BH. An interrupt was detected during the instruction ending at location 0122H. Internal RAM locations 0AH and 0BH contain the values 23H and 01H, respectively. The instruction,

RETI

will leave the Stack Pointer equal to 09H and return program execution to location 0123H.

Bytes: 1

Cycles: 2

Encoding:

0 0 1 1	0 0 1 0

Operation: RETI
$(PC_{15-8}) \leftarrow ((SP))$
$(SP) \leftarrow (SP) - 1$
$(PC_{7-0}) \leftarrow ((SP))$
$(SP) \leftarrow (SP) - 1$

RL A

Function: Rotate Accumulator Left

Description: The eight bits in the Accumulator are rotated one bit to the left. Bit 7 is rotated into the bit 0 position. No flags are affected.

Example: The Accumulator holds the value 0C5H (11000101B). The instruction,

```
RL   A
```

leaves the Accumulator holding the value 8BH (10001011B) with the carry unaffected.

Bytes: 1

Cycles: 1

Encoding:

0 0 1 0	0 0 1 1

Operation:

```
RL
(An + 1) ← (An)   n = 0 – 6
(A0) ← (A7)
```

RLC A

Function: Rotate Accumulator Left through the Carry flag

Description: The eight bits in the Accumulator and the carry flag are together rotated one bit to the left. Bit 7 moves into the carry flag; the original state of the carry flag moves into the bit 0 position. No other flags are affected.

Example: The Accumulator holds the value 0C5H (11000101B), and the carry is zero. The instruction,

```
RLC   A
```

leaves the Accumulator holding the value 8BH (10001010B) with the carry set.

Bytes: 1

Cycles: 1

Encoding:

0 0 1 1	0 0 1 1

Operation:

```
RLC
(An + 1) ← (An)   n = 0 – 6
(A0) ← (C)
(C) ← (A7)
```

RR A

Function: Rotate Accumulator Right

Description: The eight bits in the Accumulator are rotated one bit to the right. Bit 0 is rotated into the bit 7 position. No flags are affected.

Example: The Accumulator holds the value 0C5H (11000101B). The instruction,

RR A

leaves the Accumulator holding the value 0E2H (11100010B) with the carry unaffected.

Bytes: 1

Cycles: 1

Encoding:

0 0 0 0	0 0 1 1

Operation:
RR
(An) ← (A_n + 1) n = 0 − 6
(A7) ← (A0)

RRC A

Function: Rotate Accumulator Right through Carry flag

Description: The eight bits in the Accumulator and the carry flag are together rotated one bit to the right. Bit 0 moves into the carry flag; the original value of the carry flag moves into the bit 7 position. No other flags are affected.

Example: The Accumulator holds the value 0C5H (11000101B), the carry is zero. The instruction,

RRC A

leaves the Accumulator holding the value 62 (01100010B) with the carry set.

Bytes: 1

Cycles: 1

Encoding:

0 0 0 1	0 0 1 1

Operation:
RRC
(An) ← (An + 1) n = 0 − 6
(A7) ← (C)
(C) ← (A0)

SETB <bit>

Function: Set Bit

Description: SETB sets the indicated bit to one. SETB can operate on the carry flag or any directly addressable bit. No other flags are affected.

Example: The carry flag is cleared. Output Port 1 has been written with the value 34H (00110100B). The instructions,

```
SETB  C
SETB  P1.0
```

will leave the carry flag set to 1 and change the data output on Port 1 to 35H (00110101B).

SETB C

Bytes: 1

Cycles: 1

Encoding:

1 1 0 1	0 0 1 1

Operation:
SETB
(C) ← 1

SETB bit

Bytes: 2

Cycles: 1

Encoding:

1 1 0 1	0 0 1 0	bit address

Operation:
SETB
(bit) ← 1

SJMP rel

Function: Short Jump

Description: Program control branches unconditionally to the address indicated. The branch destination is computed by adding the signed displacement in the second instruction byte to the PC, after incrementing the PC twice. Therefore, the range of destinations allowed is from 128 bytes preceding this instruction to 127 bytes following it.

Example: The label "RELADR" is assigned to an instruction at program memory location 0123H. The instruction,

```
SJMP   RELADR
```

will assemble into location 0100H. After the instruction is executed, the PC will contain the value 0123H.

(*Note:* Under the above conditions the instruction following SJMP will be at 102H. Therefore, the displacement byte of the instruction will be the relative offset (0123H-0102H) = 21H. Put another way, an SJMP with a displacement of 0FEH would be a one-instruction infinite loop.)

Bytes: 2

Cycles: 2

Encoding:

1 0 0 0	0 0 0 0	rel. address

Operation: SJMP
(PC) ← (PC) + 2
(PC) ← (PC) + rel

SUBB A,<src-byte>

Function: Subtract with borrow

Description: SUBB subtracts the indicated variable and the carry flag together from the Accumulator, leaving the result in the Accumulator. SUBB sets the carry (borrow) flag if a borrow is needed for bit 7, and clears C otherwise. (If C was set *before* executing a SUBB instruction, this indicates that a borrow was needed for the previous step in a multiple precision subtraction, so the carry is subtracted from the Accumulator along with the source operand.) AC is set if a borrow is needed for bit 3, and cleared otherwise. OV is set if a borrow is needed into bit 6, but not into bit 7, or into bit 7, but not bit 6.

When subtracting signed integers OV indicates a negative number produced when a negative value is subtracted from a positive value, or a positive result when a positive number is subtracted from a negative number.

The source operand allows four addressing modes: register, direct, register-indirect, or immediate.

Example: The Accumulator holds 0C9H (11001001B), register 2 holds 54H (01010100B), and the carry flag is set. The instruction,

```
SUBB  A,R2
```

will leave the value 74H (01110100B) in the accumulator, with the carry flag and AC cleared but OV set.

Notice that 0C9H minus 54H is 75H. The difference between this and the above result is due to the carry (borrow) flag being set before the operation. If the state of the carry is not known before starting a single or multiple-precision subtraction, it should not be explicitly cleared by a CLR C instruction.

SUBB A,Rn

Bytes: 1

Cycles: 1

Encoding:

1 0 0 1	1 r r r

Operation: SUBB

(A) ← (A) − (C) − (Rn)

SUBB A,direct

Bytes: 2

Cycles: 1

Encoding:

1 0 0 1	0 1 0 1	direct address

Operation: SUBB
(A) ← (A) − (C) − (direct)

SUBB A,@Ri

Bytes: 1

Cycles: 1

Encoding:

1 0 0 1	0 1 1 i

Operation: SUBB
(A) ← (A) − (C) − ((Ri))

SUBB A,#data

Bytes: 2

Cycles: 1

Encoding:

1 0 0 1	0 1 0 0	immediate data

Operation: SUBB
(A) ← (A) − (C) − #data

SWAP A

Function: Swap nibbles within the Accumulator

Description: SWAP A interchanges the low- and high-order nibbles (four-bit fields) of the Accumulator (bits 3-0 and bits 7-4). The operation can also be thought of as a four-bit rotate instruction. No flags are affected.

Example: The Accumulator holds the value 0C5H (11000101B). The instruction,

SWAP A

leaves the Accumulator holding the value 5CH (01011100B).

Bytes: 1

Cycles: 1

Encoding:

1 1 0 0	0 1 0 0

Operation: SWAP
$(A_{3-0}) \rightleftarrows (A_{7-4})$

XCH A,<byte>

Function: Exchange Accumulator with byte variable

Description: XCH loads the Accumulator with the contents of the indicated variable, at the same time writing the original Accumulator contents to the indicated variable. The source/destination operand can use register, direct, or register-indirect addressing.

Example: R0 contains the address 20H. The Accumulator holds the value 3FH (00111111B). Internal RAM location 20H holds the value 75H (01110101B). The instruction,

XCH A,@R0

will leave RAM location 20H holding the values 3FH (00111111B) and 75H (01110101B) in the accumulator.

XCH A,Rn

Bytes: 1

Cycles: 1

Encoding:

1 1 0 0	1 r r r

Operation: XCH
(A) $\rightleftarrows$ (Rn)

XCH A,direct

Bytes: 2

Cycles: 1

Encoding:

1 1 0 0	0 1 0 1	direct address

Operation: XCH
(A) $\rightleftarrows$ (direct)

XCH A,@Ri

Bytes: 1

Cycles: 1

Encoding:

1 1 0 0	0 1 1 i

Operation: XCH
(A) $\rightleftarrows$ ((Ri))

XCHD A,@Ri

Function: Exchange Digit

Description: XCHD exchanges the low-order nibble of the Accumulator (bits 3-0), generally representing a hexadecimal or BCD digit, with that of the internal RAM location indirectly addressed by the specified register. The high-order nibbles (bits 7-4) of each register are not affected. No flags are affected.

Example: R0 contains the address 20H. The Accumulator holds the value 36H (00110110B). Internal RAM location 20H holds the value 75H (01110101B). The instruction,

XCHD A,@R0

will leave RAM location 20H holding the value 76H (01110110B) and 35H (00110101B) in the Accumulator.

Bytes: 1

Cycles: 1

Encoding:

1 1 0 1	0 1 1 i

Operation: XCHD

$(A_{3\text{-}0}) \rightleftarrows ((Ri_{3\text{-}0}))$

XRL <dest-byte>,<src-byte>

Function: Logical Exclusive-OR for byte variables

Description: XRL performs the bitwise logical Exclusive-OR operation between the indicated variables, storing the results in the destination. No flags are affected.

The two operands allow six addressing mode combinations. When the destination is the Accumulator, the source can use register, direct, register-indirect, or immediate addressing; when the destination is a direct address, the source can be the Accumulator or immediate data.

(*Note:* When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, *not* the input pins.)

Example: If the Accumulator holds 0C3H (11000011B) and register 0 holds 0AAH (10101010B) then the instruction,

XRL A,R0

will leave the Accumulator holding the value 69H (01101001B).

When the destination is a directly addressed byte, this instruction can complement combinations of bits in any RAM location or hardware register. The pattern of bits to be complemented is then determined by a mask byte, either a constant contained in the instruction or a variable computed in the Accumulator at run-time. The instruction,

XRL P1,#00110001B

will complement bits 5, 4, and 0 of output Port 1.

XRL A,Rn

Bytes: 1

Cycles: 1

Encoding:

0 1 1 0	1 r r r

Operation: XRL
(A) ← (A) ∀ (Rn)

XRL A,direct

Bytes: 2

Cycles: 1

Encoding:

0 1 1 0	0 1 0 1	direct address

Operation: XRL
(A) ← (A) ∀ (direct)

XRL A,@Ri

Bytes: 1

Cycles: 1

Encoding:

0 1 1 0	0 1 1 i

Operation: XRL
(A) ← (A) ∀ ((Ri))

XRL A,#data

Bytes: 2

Cycles: 1

Encoding:

0 1 1 0	0 1 0 0	immediate data

Operation: XRL
(A) ← (A) ∀ #data

XRL direct,A

Bytes: 2

Cycles: 1

Encoding:

0 1 1 0	0 0 1 0	direct address

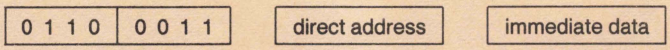
Operation: XRL
(direct) ← (direct) ∀ (A)

XRL direct,#data

Bytes: 3

Cycles: 2

Encoding:



Operation: XRL
(direct) ← (direct) ∀ #data

XRL dirext,#data

Bytes: 3

Cycles: 2

Encoding: 0 1 1 0 0 0 1 1 | direct address | immediate data

Operation: XRL

(direct) ← (direct) ∨ #data

MCS®-51 Data Sheets

8

MCS®-51
8-BIT CONTROL-ORIENTED MICROCOMPUTERS
8031/8051
8031AH/8051AH
8032AH/8052AH
8751H/8751H-12/8751H-88

- High Performance HMOS Process
- Internal Timers/Event Counters
- 2-Level Interrupt Priority Structure
- 32 I/O Lines (Four 8-Bit Ports)
- 64K Program Memory Space
- Security Feature Protects EPROM Parts Against Software Piracy
- Boolean Processor
- Bit-Addressable RAM
- Programmable Full Duplex Serial Channel
- 111 Instructions (64 Single-Cycle)
- 64K Data Memory Space

The MCS®-51 products are optimized for control applications. Byte-processing and numerical operations on small data structures are facilitated by a variety of fast addressing modes for accessing the internal RAM. The instruction set provides a convenient menu of 8-bit arithmetic instructions, including multiply and divide instructions. Extensive on-chip support is provided for one-bit variables as a separate data type, allowing direct bit manipulation and testing in control and logic systems that require Boolean processing.

Device	Internal Memory		Timers/ Event Counters	Interrupts
	Program	Data		
8052AH	8K x 8 ROM	256 x 8 RAM	3 x 16-Bit	6
8051AH	4K x 8 ROM	128 x 8 RAM	2 x 16-Bit	5
8051	4K x 8 ROM	128 x 8 RAM	2 x 16-Bit	5
8032AH	none	256 x 8 RAM	3 x 16-Bit	6
8031AH	none	128 x 8 RAM	2 x 16-Bit	5
8031	none	128 x 8 RAM	2 x 16-Bit	5
8751H	4K x 8 EPROM	128 x 8 RAM	2 x 16-Bit	5
8751H-12	4K x 8 EPROM	128 x 8 RAM	2 x 16-Bit	5
8751H-88	4K x 8 EPROM	128 x 8 RAM	2 x 16-Bit	5

The 8751H is an EPROM version of the 8051AH; that is, the on-chip Program Memory can be electrically programmed, and can be erased by exposure to ultraviolet light. It is fully compatible with its predecessor, the 8751-8, but incorporates two new features: a Program Memory Security bit that can be used to protect the EPROM against unauthorized read-out, and a programmable baud rate modification bit (SMOD). SMOD is not present in the 8751H-12 or the 8751H-88. The 8751H-88 also only operates up to 8 MHz.

November 1986
Order Number: 270048-002

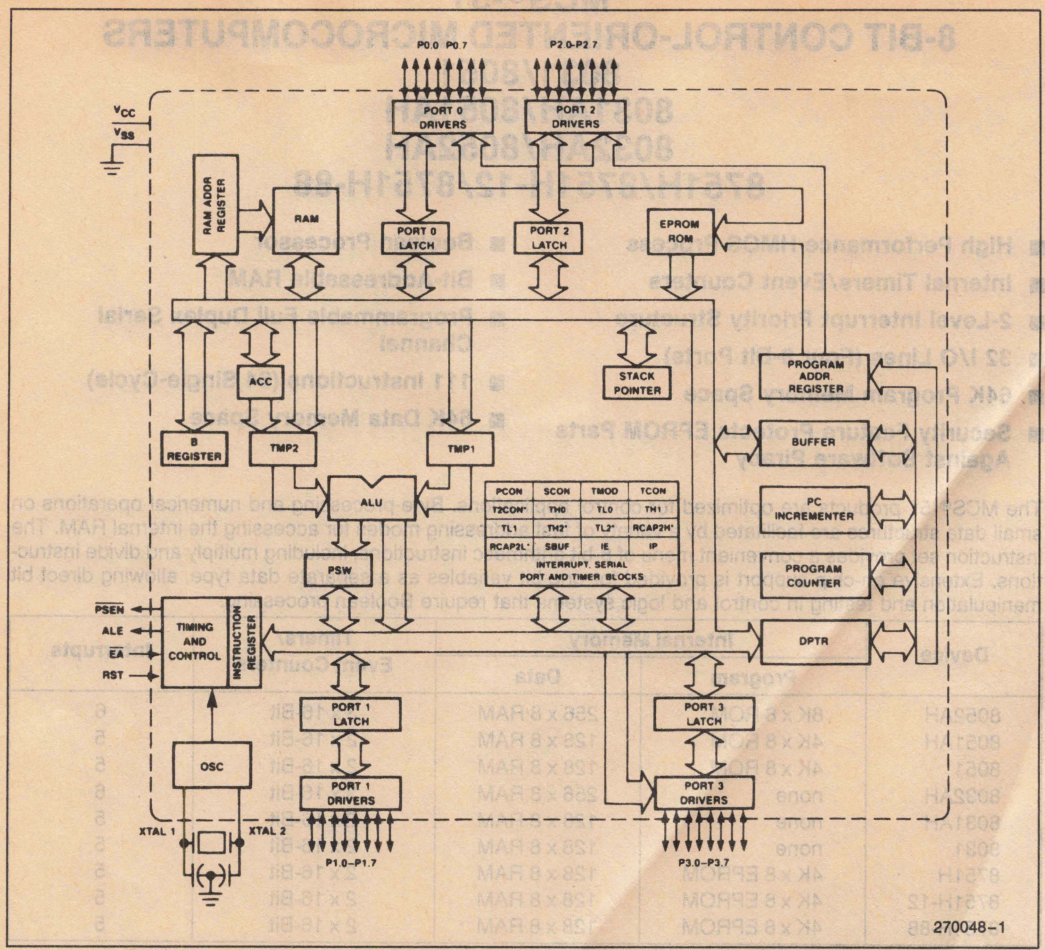


Figure 1. MCS®-51 Block Diagram

PIN DESCRIPTIONS

VCC

Supply voltage.

VSS

Circuit ground.

Port 0

Port 0 is an 8-bit open drain bidirectional I/O port. As an output port each pin can sink 8 LS TTL inputs. Port 0 pins that have 1s written to them float, and in that state can be used as high-impedance inputs.

Port 0 is also the multiplexed low-order address and data bus during accesses to external Program and Data Memory. In this application it uses strong internal pullups when emitting 1s and can source and sink 8 LS TTL inputs.

Port 0 also receives the code bytes during programming of the EPROM parts, and outputs the code bytes during program verification of the ROM and EPROM parts. External pullups are required during program verification.

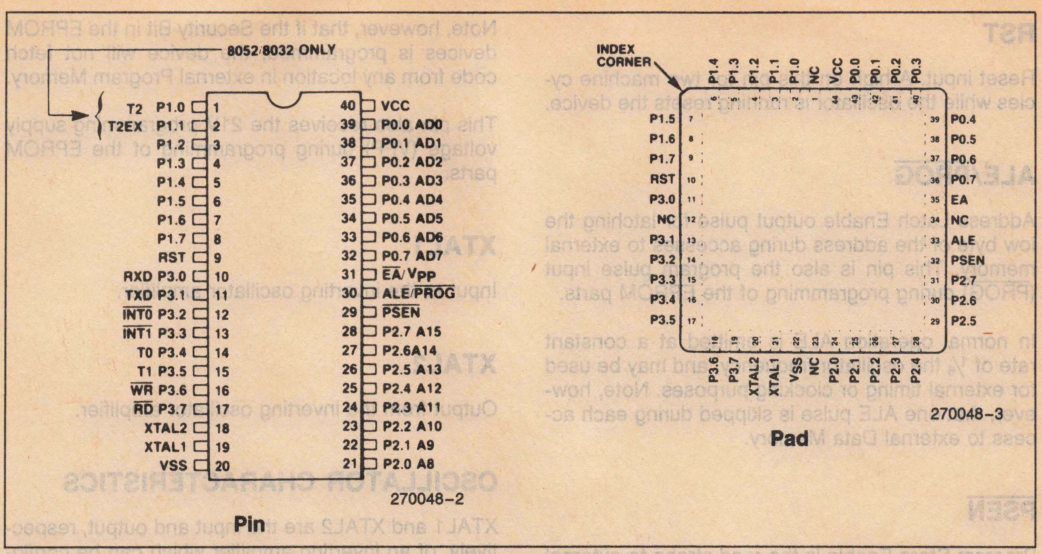


Figure 2. MCS®-51 Connections

Port 1

Port 1 is an 8-bit bidirectional I/O port with internal pullups. The Port 1 output buffers can sink/source 4 LS TTL inputs. Port 1 pins that have 1s written to them are pulled high by the internal pullups, and in that state can be used as inputs. As inputs, Port 1 pins that are externally being pulled low will source current (IIL on the data sheet) because of the internal pullups.

Port 1 also receives the low-order address bytes during programming of the EPROM parts and during program verification of the ROM and EPROM parts.

In the 8032AH and 8052AH, Port 1 pins P1.0 and P1.1 also serve the T2 and T2EX functions, respectively.

Port 2

Port 2 is an 8-bit bidirectional I/O port with internal pullups. The Port 2 output buffers can sink/source 4 LS TTL inputs. Port 2 pins that have 1s written to them are pulled high by the internal pullups, and in that state can be used as inputs. As inputs, Port 2 pins that are externally being pulled low will source current (IIL on the data sheet) because of the internal pullups.

Port 2 emits the high-order address byte during fetches from external Program Memory and during accesses to external Data Memory that use 16-bit addresses (MOVX @DPTR). In this application it uses strong internal pullups when emitting 1s. During accesses to external Data Memory that use 8-bit addresses (MOVX @Ri), Port 2 emits the contents of the P2 Special Function Register.

Port 2 also receives the high-order address bits during programming of the EPROM parts and during program verification of the ROM and EPROM parts.

Port 3

Port 3 is an 8-bit bidirectional I/O port with internal pullups. The Port 3 output buffers can sink/source 4 LS TTL inputs. Port 3 pins that have 1s written to them are pulled high by the internal pullups, and in that state can be used as inputs. As inputs, Port 3 pins that are externally being pulled low will source current (IIL on the data sheet) because of the pullups.

Port 3 also serves the functions of various special features of the MCS-51 Family, as listed below:

Port Pin	Alternative Function
P3.0	RXD (serial input port)
P3.1	TXD (serial output port)
P3.2	$\overline{INT0}$ (external interrupt 0)
P3.3	$\overline{INT1}$ (external interrupt 1)
P3.4	T0 (Timer 0 external input)
P3.5	T1 (Timer 1 external input)
P3.6	$\overline{WR}$ (external data memory write strobe)
P3.7	$\overline{RD}$ (external data memory read strobe)

RST

Reset input. A high on this pin for two machine cycles while the oscillator is running resets the device.

ALE/$\overline{PROG}$

Address Latch Enable output pulse for latching the low byte of the address during accesses to external memory. This pin is also the program pulse input ($\overline{PROG}$) during programming of the EPROM parts.

In normal operation ALE is emitted at a constant rate of 1/6 the oscillator frequency, and may be used for external timing or clocking purposes. Note, however, that one ALE pulse is skipped during each access to external Data Memory.

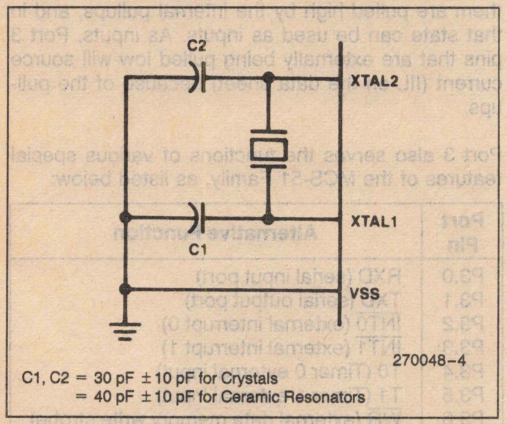
$\overline{PSEN}$

Program Store Enable is the read strobe to external Program Memory.

When the device is executing code from external Program Memory, $\overline{PSEN}$ is activated twice each machine cycle, except that two $\overline{PSEN}$ activations are skipped during each access to external Data Memory.

$\overline{EA}$/VPP

External Access enable $\overline{EA}$ must be strapped to VSS in order to enable any MCS-51 device to fetch code from external Program memory locations 0 to 0FFFH (0 to 1FFFH, in the 8032AH and 8052AH).



C1, C2 = 30 pF ±10 pF for Crystals
= 40 pF ±10 pF for Ceramic Resonators

Figure 3. Oscillator Connections

Note, however, that if the Security Bit in the EPROM devices is programmed, the device will not fetch code from any location in external Program Memory.

This pin also receives the 21V programming supply voltage (VPP) during programming of the EPROM parts.

XTAL1

Input to the inverting oscillator amplifier.

XTAL2

Output from the inverting oscillator amplifier.

OSCILLATOR CHARACTERISTICS

XTAL1 and XTAL2 are the input and output, respectively, of an inverting amplifier which can be configured for use as an on-chip oscillator, as shown in Figure 3. Either a quartz crystal or ceramic resonator may be used. More detailed information concerning the use of the on-chip oscillator is available in Application Note AP-155, "Oscillators for Microcontrollers."

To drive the device from an external clock source, XTAL1 should be grounded, while XTAL2 is driven, as shown in Figure 4. There are no requirements on the duty cycle of the external clock signal, since the input to the internal clocking circuitry is through a divide-by-two flip-flop, but minimum and maximum high and low times specified on the Data Sheet must be observed.

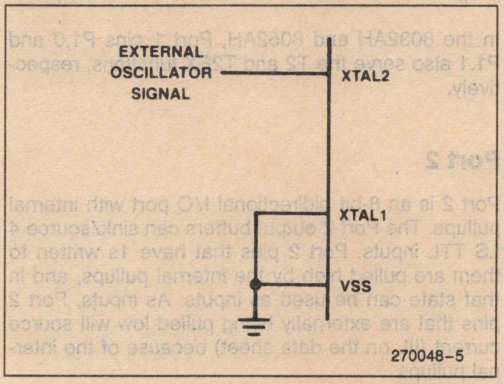


Figure 4. External Drive Configuration

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias .....0°C to 70°C
Storage Temperature .....−65°C to +150°C
Voltage on $\overline{EA}$/VPP Pin to VSS ..−0.5V to +21.5V
Voltage on Any Other Pin to VSS....−0.5V to +7V
Power Dissipation.....1.5W

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS T_A = 0°C to 70°C; VCC = 5V ±10%; VSS = 0V

Symbol	Parameter	Min	Max	Units	Test Conditions
VIL	Input Low Voltage (Except $\overline{EA}$ Pin of 8751H, 8751H-12 & 8751H-88)	−0.5	0.8	V	
VIL1	Input Low Voltage to $\overline{EA}$ Pin of 8751H, 8751H-12 & 8751H-88	0	0.7	V	
VIH	Input High Voltage (Except XTAL2, RST)	2.0	VCC + 0.5	V	
VIH1	Input High Voltage to XTAL2, RST	2.5	VCC + 0.5	V	XTAL1 = VSS
VOL	Output Low Voltage (Ports 1, 2, 3)*		0.45	V	IOL = 1.6 mA
VOL1	Output Low Voltage (Port 0, ALE, $\overline{PSEN}$)*				
	8751H, 8751H-12 & 8751H-88		0.60 0.45	V V	IOL = 3.2 mA IOL = 2.4 mA
	All Others		0.45	V	IOL = 3.2 mA
VOH	Output High Voltage (Ports 1, 2, 3, ALE, $\overline{PSEN}$)	2.4		V	IOH = −80 µA
VOH1	Output High Voltage (Port 0 in External Bus Mode)	2.4		V	IOH = −400 µA
IIL	Logical 0 Input Current (Ports 1, 2, 3, RST) 8032AH, 8052AH All Others		−800 −500	µA µA	Vin = 0.45V Vin = 0.45V
IIL1	Logical 0 Input Current to $\overline{ES}$ Pin of 8751H, 8751H-12, & 8751H-88 Only		−15	mA	
IIL2	Logical 0 Input Current (XTAL2)		−3.2	mA	Vin = 0.45V
ILI	Input Leakage Current (Port 0) 8751H, 8751H-12 & 8751H-88 All Others		 ±100 ±10	 µA µA	 0.45 ≤ Vin ≤ VCC 0.45 ≤ Vin ≤ VCC
IIH	Logical 1 Input Current to $\overline{EA}$ Pin of 8751H, 8751H-12 & 8751H-88		500	µA	
IIH1	Input Current to RST to Activate Reset		500	µA	Vin < (VCC − 1.5V)
ICC	Power Supply Current: 8031/8051 8031AH/8051AH 8032AH/8052AH 8751H/8751H-12/8751H-88		 160 125 175 250	 mA mA mA mA	 All Outputs Disconnected; $\overline{EA}$ = VCC
CIO	Pin Capacitance		10	pF	Test freq = 1 MHz

***NOTE:**
Capacitive loading on Ports 0 and 2 may cause spurious noise pulses to be superimposed on the VOLs of ALE and Ports 1 and 3. The noise is due to external bus capacitance discharging into the Port 0 and Port 2 pins when these pins make 1-to-0 transitions during bus operations. In the worst cases (capacitive loading > 100 pF), the noise pulse on the ALE line may exceed 0.8V. In such cases it may be desirable to qualify ALE with a Schmitt Trigger, or use an address latch with a Schmitt Trigger STROBE input.

A.C. CHARACTERISTICS

T_A = 0°C to +70°C; VCC = 5V ±10%; VSS = 0V;
Load Capacitance for Port 0, ALE, and PSEN = 100 pF;
Load Capacitance for All Other Outputs = 80 pF

Symbol	Parameter	12 MHz Oscillator		Variable Oscillator		Units
		Min	Max	Min	Max	
1/TCLCL	Oscillator Frequency			3.5	12.0	MHz
TLHLL	ALE Pulse Width	127		2TCLCL−40		ns
TAVLL	Address Valid to ALE Low	43		TCLCL−40		ns
TLLAX	Address Hold after ALE Low	48		TCLCL−35		ns
TLLIV	ALE Low to Valid Instr In 8751H, 8751H-12 All Others		 183 233		 4TCLCL−150 4TCLCL−100	 ns ns
TLLPL	ALE Low to $\overline{PSEN}$ Low	58		TCLCL−25		ns
TPLPH	$\overline{PSEN}$ Pulse Width 8751H, 8751H-12 All Others	 190 215		 3TCLCL−60 3TCLCL−35		 ns ns
TPLIV	$\overline{PSEN}$ Low to Valid Instr In 8751H, 8751H-12 All Others		 100 125		 3TCLCL−150 3TCLCL−125	 ns ns
TPXIX	Input Instr Hold after $\overline{PSEN}$	0		0		ns
TPXIZ	Input Instr Float after $\overline{PSEN}$		63		TCLCL−20	ns
TPXAV	$\overline{PSEN}$ to Address Valid	75		TCLCL−8		ns
TAVIV	Address to Valid Instr In 8751H, 8751H-12 All Others		 267 302		 5TCLCL−150 5TCLCL−115	 ns ns
TPLAZ	$\overline{PSEN}$ Low to Address Float		20		20	ns
TRLRH	$\overline{RD}$ Pulse Width	400		6TCLCL−100		ns
TWLWH	$\overline{WR}$ Pulse Width	400		6TCLCL−100		ns
TRLDV	$\overline{RD}$ Low to Valid Data In		252		5TCLCL−165	ns
TRHDX	Data Hold after $\overline{RD}$	0		0		ns
TRHDZ	Data Float after $\overline{RD}$		97		2TCLCL−70	ns
TLLDV	ALE Low to Valid Data In		517		8TCLCL−150	ns
TAVDV	Address to Valid Data In		585		9TCLCL−165	ns
TLLWL	ALE Low to $\overline{RD}$ or $\overline{WR}$ Low	200	300	3TCLCL−50	3TCLCL+50	ns
TAVWL	Address to $\overline{RD}$ or $\overline{WR}$ Low	203		4TCLCL−130		ns
TQVWX	Data Valid to $\overline{WR}$ Transition 8751H, 8751H-12 All Others	 13 23		 TCLCL−70 TCLCL−60		 ns ns
TQVWH	Data Valid to $\overline{WR}$ High	433		7TCLCL−150		ns
TWHQX	Data Hold after $\overline{WR}$	33		TCLCL−50		ns
TRLAZ	$\overline{RD}$ Low to Address Float		20		20	ns
TWHLH	$\overline{RD}$ or $\overline{WR}$ High to ALE High 8751H, 8751H-12 All Others	 33 43	 133 123	 TCLCL−50 TCLCL−40	 TCLCL+50 TCLCL+40	 ns ns

NOTE:
*This table does not include the 8751-88 A.C. characteristics (see next page).

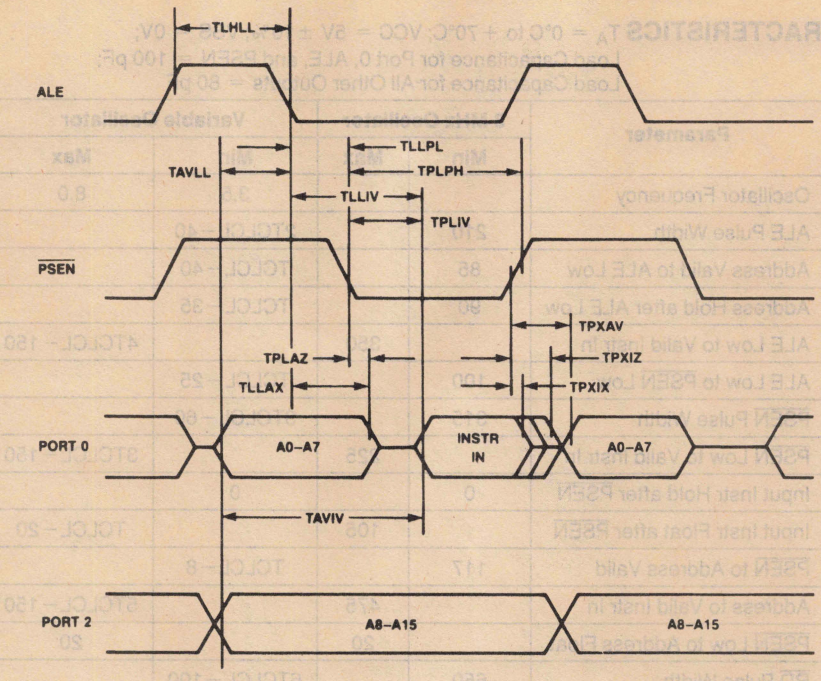
This Table is only for the 8751H-88

A.C. CHARACTERISTICS

T_A = 0°C to +70°C; VCC = 5V ±10%; VSS = 0V;
Load Capacitance for Port 0, ALE, and PSEN = 100 pF;
Load Capacitance for All Other Outputs = 80 pF

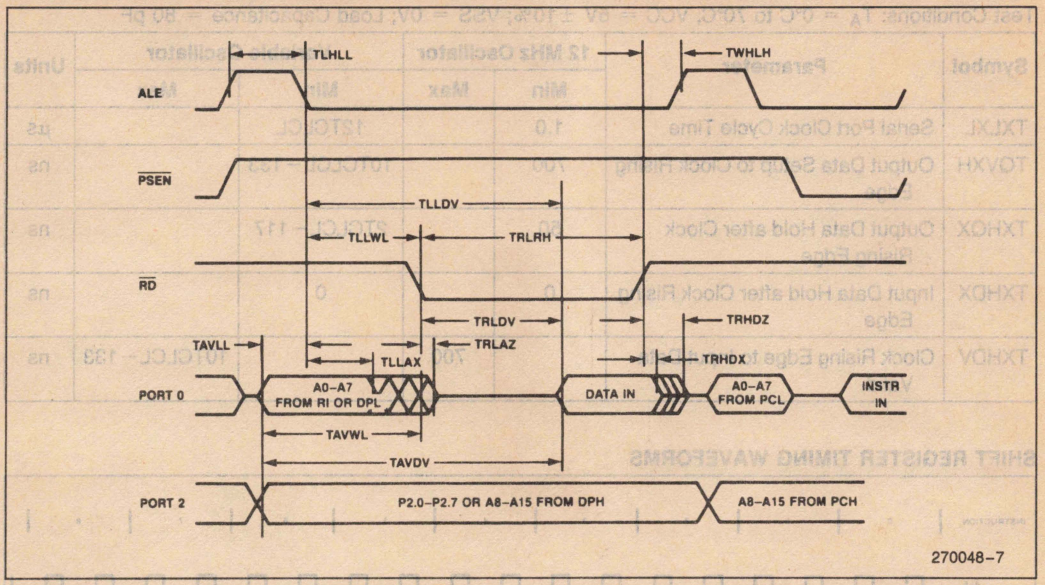
Symbol	Parameter	8 MHz Oscillator		Variable Oscillator		Units
		Min	Max	Min	Max	
1/TCLCL	Oscillator Frequency			3.5	8.0	MHz
TLHLL	ALE Pulse Width	210		2TCLCL−40		ns
TAVLL	Address Valid to ALE Low	85		TCLCL−40		ns
TLLAX	Address Hold after ALE Low	90		TCLCL−35		ns
TLLIV	ALE Low to Valid Instr In		350		4TCLCL−150	ns
TLLPL	ALE Low to $\overline{PSEN}$ Low	100		TCLCL−25		ns
TPLPH	$\overline{PSEN}$ Pulse Width	315		3TCLCL−60		ns
TPLIV	$\overline{PSEN}$ Low to Valid Instr In		225		3TCLCL−150	ns
TPXIX	Input Instr Hold after $\overline{PSEN}$	0		0		ns
TPXIZ	Input Instr Float after $\overline{PSEN}$		105		TCLCL−20	ns
TPXAV	$\overline{PSEN}$ to Address Valid	117		TCLCL−8		ns
TAVIV	Address to Valid Instr In		475		5TCLCL−150	ns
TPLAZ	$\overline{PSEN}$ Low to Address Float		20		20	ns
TRLRH	$\overline{RD}$ Pulse Width	650		6TCLCL−100		ns
TWLWH	$\overline{WR}$ Pulse Width	650		6TCLCL−100		ns
TRLDV	$\overline{RD}$ Low to Valid Data In		460		5TCLCL−165	ns
TRHDX	Data Hold after $\overline{RD}$	0		0		ns
TRHDZ	Data Float after $\overline{RD}$		180		2TCLCL−70	ns
TLLDV	ALE Low to Valid Data In		850		8TCLCL−150	ns
TAVDV	Address to Valid Data In		960		9TCLCL−165	ns
TLLWL	ALE Low to $\overline{RD}$ or $\overline{WR}$ Low	325	425	3TCLCL−50	3TCLCL+50	ns
TAVWL	Address to $\overline{RD}$ or $\overline{WR}$ Low	370		4TCLCL−130		ns
TQVWX	Data Valid to $\overline{WR}$ Transition	55		TCLCL−70		ns
TQVWH	Data Valid to $\overline{WR}$ High	725		7TCLCL−150		ns
TWHQX	Data Hold after $\overline{WR}$	75		TCLCL−50		ns
TRLAZ	$\overline{RD}$ Low to Address Float		20		20	ns
TWHLH	$\overline{RD}$ or $\overline{WR}$ High to ALE High	75	175	TCLCL−50	TCLCL+50	ns

EXTERNAL PROGRAM MEMORY READ CYCLE

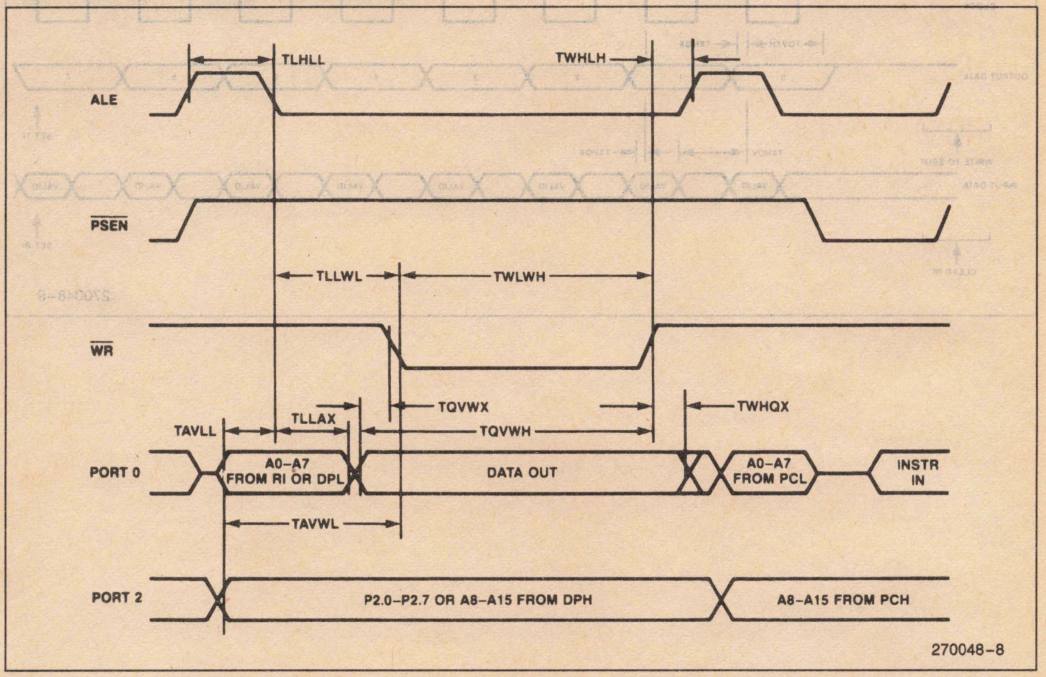


270048–6

EXTERNAL DATA MEMORY READ CYCLE



EXTERNAL DATA MEMORY WRITE CYCLE

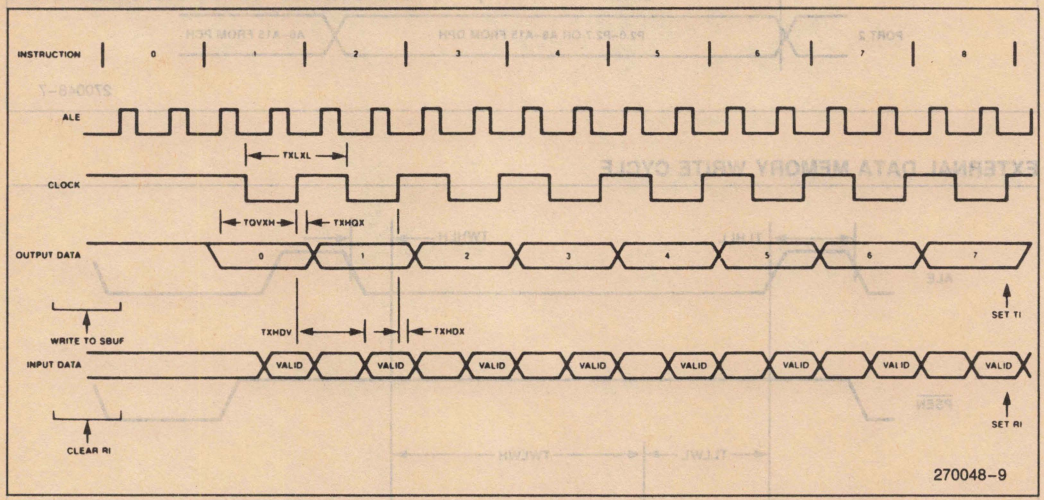


SERIAL PORT TIMING—SHIFT REGISTER MODE

Test Conditions: T_A = 0°C to 70°C; VCC = 5V ±10%; VSS = 0V; Load Capacitance = 80 pF

Symbol	Parameter	12 MHz Oscillator		Variable Oscillator		Units
		Min	Max	Min	Max	
TXLXL	Serial Port Clock Cycle Time	1.0		12TCLCL		μs
TQVXH	Output Data Setup to Clock Rising Edge	700		10TCLCL−133		ns
TXHQX	Output Data Hold after Clock Rising Edge	50		2TCLCL−117		ns
TXHDX	Input Data Hold after Clock Rising Edge	0		0		ns
TXHDV	Clock Rising Edge to Input Data Valid		700		10TCLCL−133	ns

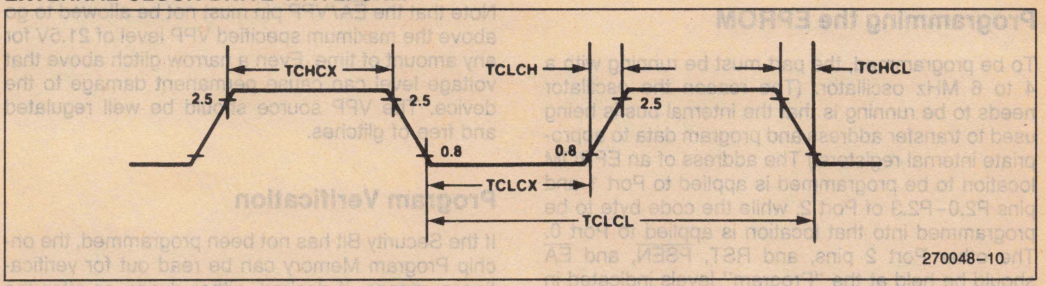
SHIFT REGISTER TIMING WAVEFORMS



EXTERNAL CLOCK DRIVE

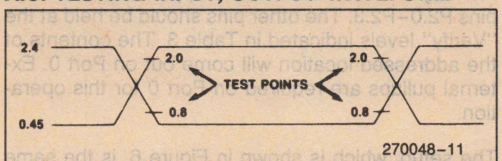
Symbol	Parameter	Min	Max	Units
1/TCLCL	Oscillator Frequency (except 8751H-88)	3.5	12	MHz
	8751H-88	3.5	8	MHz
TCHCX	High Time	20		ns
TCLCX	Low Time	20		ns
TCLCH	Rise Time		20	ns
TCHCL	Fall Time		20	ns

EXTERNAL CLOCK DRIVE WAVEFORM



270048-10

A.C. TESTING INPUT, OUTPUT WAVEFORM



270048-11

A.C. Testing: Inputs are driven at 2.4V for a Logic "1" and 0.45V for a Logic "0". Timing measurements are made at 2.0V for a Logic "1" and 0.8V for a Logic "0".

EPROM CHARACTERISTICS

Table 3. EPROM Programming Modes

Mode	RST	PSEN	ALE	EA	P2.7	P2.6	P2.5	P2.4
Program	1	0	0*	VPP	1	0	X	X
Inhibit	1	0	1	X	1	0	X	X
Verify	1	0	1	1	0	0	X	X
Security Set	1	0	0*	VPP	1	1	X	X

NOTE:
"1" = logic high for that pin
"0" = logic low for that pin
"X" = "don't care"
"VPP" = +21V ±0.5V
*ALE is pulsed low for 50 ms.

Programming the EPROM

To be programmed, the part must be running with a 4 to 6 MHz oscillator. (The reason the oscillator needs to be running is that the internal bus is being used to transfer address and program data to appropriate internal registers.) The address of an EPROM location to be programmed is applied to Port 1 and pins P2.0–P2.3 of Port 2, while the code byte to be programmed into that location is applied to Port 0. The other Port 2 pins, and RST, PSEN, and EA should be held at the "Program" levels indicated in Table 3. ALE is **pulsed** low for 50 ms to program the code byte into the addressed EPROM location. The setup is shown in Figure 5.

Normally EA is held at a logic high until just before ALE is to be pulsed. Then EA is raised to +21V, ALE is pulsed, and then EA is returned to a logic high. Waveforms and detailed timing specifications are shown in later sections of this data sheet.

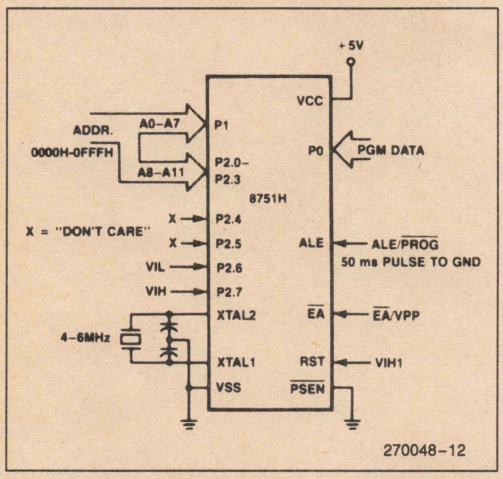


Figure 5. Programming Configuration

Note that the EA/VPP pin must not be allowed to go above the maximum specified VPP level of 21.5V for any amount of time. Even a narrow glitch above that voltage level can cause permanent damage to the device. The VPP source should be well regulated and free of glitches.

Program Verification

If the Security Bit has not been programmed, the on-chip Program Memory can be read out for verification purposes, if desired, either during or after the programming operation. The address of the Program Memory location to be read is applied to Port 1 and pins P2.0–P2.3. The other pins should be held at the "Verify" levels indicated in Table 3. The contents of the addressed location will come out on Port 0. External pullups are required on Port 0 for this operation.

The setup, which is shown in Figure 6, is the same as for programming the EPROM except that pin P2.7 is held at a logic low, or may be used as an active-low read strobe.

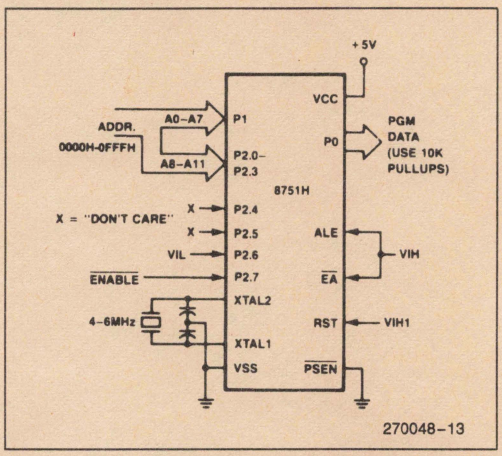


Figure 6. Program Verification

EPROM Security

The security feature consists of a "locking" bit which when programmed denies electrical access by any external means to the on-chip Program Memory. The bit is programmed as shown in Figure 7. The setup and procedure are the same as for normal EPROM programming, except that P2.6 is held at a logic high. Port 0, Port 1, and pins P2.0–P2.3 may be in any state. The other pins should be held at the "Security" levels indicated in Table 3.

Once the Security Bit has been programmed, it can be cleared only by full erasure of the Program Memory. While it is programmed, the internal Program Memory can not be read out, the device can not be further programmed, and it **can not execute out of external program memory**. Erasing the EPROM, thus clearing the Security Bit, restores the device's full functionality. It can then be reprogrammed.

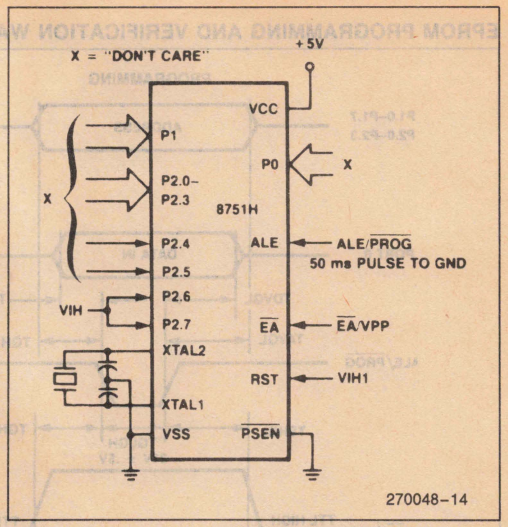


Figure 7. Programming the Security Bit

Erasure Characteristics

Erasure of the EPROM begins to occur when the chip is exposed to light with wavelengths shorter than approximately 4,000 Angstroms. Since sunlight and fluorescent lighting have wavelengths in this range, exposure to these light sources over an extended time (about 1 week in sunlight, or 3 years in room-level fluorescent lighting) could cause inadvertent erasure. If an application subjects the device to this type of exposure, it is suggested that an opaque label be placed over the window.

The recommended erasure procedure is exposure to ultraviolet light (at 2537 Angstroms) to an integrated dose of at least 15 W-sec/cm^2. Exposing the EPROM to an ultraviolet lamp of 12,000 μW/cm^2 rating for 20 to 30 minutes, at a distance of about 1 inch, should be sufficient.

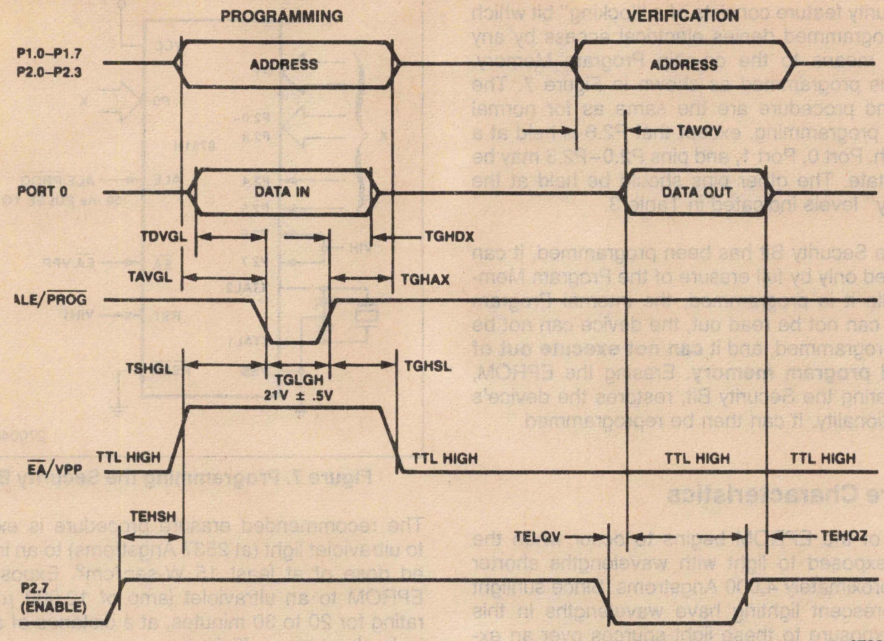
Erasure leaves the array in an all 1s state.

EPROM PROGRAMMING AND VERIFICATION CHARACTERISTICS

T_A = 21°C to 27°C; VCC = 5V ±10%; VSS = 0V

Symbol	Parameter	Min	Max	Units
VPP	Programming Supply Voltage	20.5	21.5	V
IPP	Programming Supply Current		30	mA
1/TCLCL	Oscillator Frequency	4	6	MHz
TAVGL	Address Setup to $\overline{\text{PROG}}$ Low	48TCLCL		
TGHAX	Address Hold after $\overline{\text{PROG}}$	48TCLCL		
TDVGL	Data Setup to $\overline{\text{PROG}}$ Low	48TCLCL		
TGHDX	Data Hold after $\overline{\text{PROG}}$	48TCLCL		
TEHSH	P2.7 ($\overline{\text{ENABLE}}$) High to VPP	48TCLCL		
TSHGL	VPP Setup to $\overline{\text{PROG}}$ Low	10		μs
TGHSL	VPP Hold after $\overline{\text{PROG}}$	10		μs
TGLGH	$\overline{\text{PROG}}$ Width	45	55	ms
TAVQV	Address to Data Valid		48TCLCL	
TELQV	$\overline{\text{ENABLE}}$ Low to Data Valid		48TCLCL	
TEHQZ	Data Float after $\overline{\text{ENABLE}}$	0	48TCLCL	

EPROM PROGRAMMING AND VERIFICATION WAVEFORMS

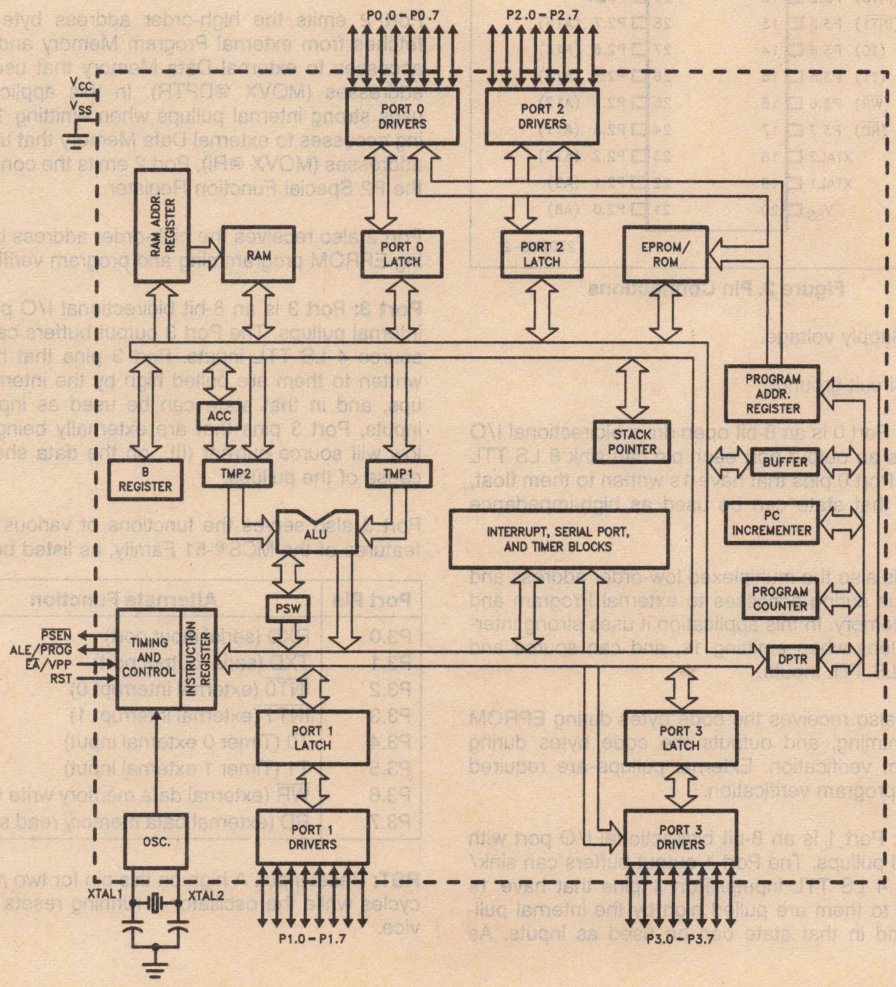


For programming conditions see Figure 5.

For verification conditions see Figure 6.

8751BH
SINGLE-CHIP 8-BIT MICROCOMPUTER WITH 4K BYTES OF EPROM PROGRAM MEMORY

- Program Memory Lock
- 128 Bytes Data Ram
- Quick Pulse Programming™ Algorithm
- 12.75 Volt Programming Voltage
- Boolean Processor
- 32 Programmable I/O Lines
- Two 16-Bit Timer/Counters
- 5 Interrupt Sources
- Programmable Serial Channel
- Separate Transmit/Receive Baud Rate Capability
- 64K External Program Memory Space
- 64K External Data Memory Space



270248-1

Figure 1. 8751BH Block Diagram

November 1986
Order Number: 270248-001

PIN DESCRIPTIONS

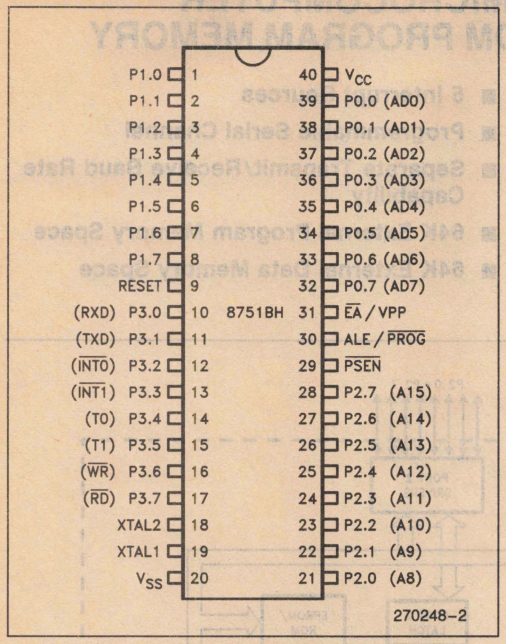


Figure 2. Pin Connections

VCC: Supply voltage.

VSS: Circuit ground.

Port 0: Port 0 is an 8-bit open drain bidirectional I/O port. As an output port each pin can sink 8 LS TTL inputs. Port 0 pins that have 1s written to them float, and in that state can be used as high-impedance inputs.

Port 0 is also the multiplexed low-order address and data bus during accesses to external Program and Data Memory. In this application it uses strong internal pullups when emitting 1s, and can source and sink 8 LS TTL inputs.

Port 0 also receives the code bytes during EPROM programming, and outputs the code bytes during program verification. External pullups are required during program verification.

Port 1: Port 1 is an 8-bit bidirectional I/O port with internal pullups. The Port 1 output buffers can sink/source 4 LS TTL inputs. Port 1 pins that have 1s written to them are pulled high by the internal pullups, and in that state can be used as inputs. As inputs, Port 1 pins that are externally being pulled low will source current (IIL, on the data sheet) because of the internal pullups.

Port 1 also receives the low-order address bytes during EPROM programming and program verification.

Port 2: Port 2 is an 8-bit bidirectional I/O port with internal pullups. The Port 2 output buffers can sink/source 4 LS TTL inputs. Port 2 pins that have 1s written to them are pulled high by the internal pullups, and in that state can be used as inputs. As inputs, Port 2 pins that are externally being pulled low will source current (IIL, on the data sheet) because of the internal pullups.

Port 2 emits the high-order address byte during fetches from external Program Memory and during accesses to external Data Memory that use 16-bit addresses (MOVX @DPTR). In this application it uses strong internal pullups when emitting 1s. During accesses to external Data Memory that use 8-bit addresses (MOVX @Ri), Port 2 emits the contents of the P2 Special Function Register.

Port 2 also receives the high-order address bits during EPROM programming and program verification.

Port 3: Port 3 is an 8-bit bidirectional I/O port with internal pullups. The Port 3 output buffers can sink/source 4 LS TTL inputs. Port 3 pins that have 1s written to them are pulled high by the internal pullups, and in that state can be used as inputs. As inputs, Port 3 pins that are externally being pulled low will source current (IIL, on the data sheet) because of the pullups.

Port 3 also serves the functions of various special features of the MCS®-51 Family, as listed below:

Port Pin	Alternate Function
P3.0	RXD (serial input port)
P3.1	TXD (serial output port)
P3.2	$\overline{INT0}$ (external interrupt 0)
P3.3	$\overline{INT1}$ (external interrupt 1)
P3.4	T0 (Timer 0 external input)
P3.5	T1 (Timer 1 external input)
P3.6	$\overline{WR}$ (external data memory write strobe)
P3.7	$\overline{RD}$ (external data memory read strobe)

RST: Reset input. A high on this pin for two machine cycles while the oscillator is running resets the device.

ALE/$\overline{PROG}$: Address Latch Enable output pulse for latching the low byte of the address during accesses to external memory. This pin is also the program pulse input (PROG) during EPROM programming.

In normal operation ALE is emitted at a constant rate of 1/6 the oscillator frequency, and may be used for external timing or clocking purposes. Note, however, that one ALE pulse is skipped during each access to external Data Memory.

$\overline{PSEN}$: Program Store Enable is the Read strobe to External Program Memory.

When the 8751BH is executing code from external Program Memory, $\overline{PSEN}$ is activated twice each machine cycle, except that two $\overline{PSEN}$ activations are skipped during each access to External Data Memory.

$\overline{EA}$/VPP: External Access enable. $\overline{EA}$ must be strapped to V_{SS} in order to enable the device to fetch code from External Program Memory locations 0000H to 0FFFH. Note, however, that if either of the Lock Bits are programmed, $\overline{EA}$ will be internally latched on reset.

$\overline{EA}$ should be strapped to VCC for internal program executions.

This pin also receives the 12.75V programming supply voltage (VPP) during EPROM programming.

XTAL1: Input to the inverting oscillator amplifier.

XTAL2: Output from the inverting oscillator amplifier.

OSCILLATOR CHARACTERISTICS

XTAL1 and XTAL2 are the input and output, respectively, of an inverting amplifier which can be configured for use as an on-chip oscillator, as shown in Figure 3. Either a quartz crystal or ceramic resonator may be used. More detailed information concerning the use of the on-chip oscillator is available in Applications Note AP-155, "Oscillators for Microcontrollers."

To drive the device from an external clock source, XTAL1 should be grounded, while XTAL2 is driven, as shown in Figure 4. There are no requirements on the duty cycle of the external clock signal, since the input to the internal clocking circuitry is through a divide-by-two flip-flop, but minimum and maximum high and low times specified on the Data Sheet must be observed.

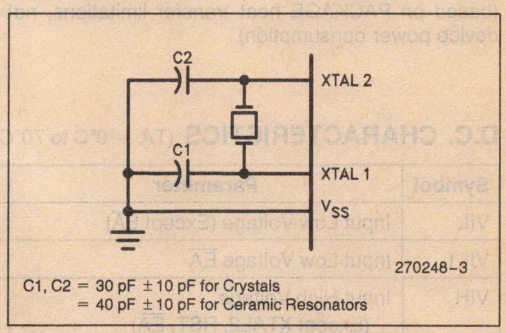


Figure 3. Oscillator Connections

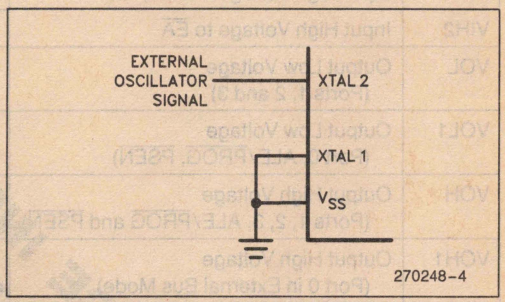


Figure 4. External Clock Drive Configuration

DESIGN CONSIDERATIONS

Exposure to light when the device is in operation may cause logic errors. For this reason, it is suggested that an opaque label be placed over the window when the die is exposed to ambient light.

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias .....0°C to 70°C
Storage Temperature .....−65°C to +150°C
Voltage on $\overline{EA}$/VPP Pin to VSS ..−0.5V to +13.0V
Voltage on Any Other Pin to VSS....−0.5V to +7V
Power Dissipation.....1.5W
(based on PACKAGE heat transfer limitations, not device power consumption)

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

NOTICE: Specifications contained within the following tables are subject to change.

D.C. CHARACTERISTICS (TA = 0°C to 70°C; VCC = 5V ±10%; VSS = 0V)

Symbol	Parameter	Min	Max	Unit	Test Conditions
VIL	Input Low Voltage (Except $\overline{EA}$)	−0.5	0.8	V	
VIL1	Input Low Voltage $\overline{EA}$	VSS	0.7	V	
VIH	Input High Voltage (Except XTAL2, RST, $\overline{EA}$)	2.0	VCC+0.5	V	
VIH1	Input High Voltage XTAL2, RST	2.5	VCC+0.5	V	XTAL1 = VSS
VIH2	Input High Voltage to $\overline{EA}$	4.5	5.5	V	
VOL	Output Low Voltage (Ports 1, 2 and 3)		0.45	V	IOL = 1.6 mA (Note 1)
VOL1	Output Low Voltage (Port 0, ALE/$\overline{PROG}$, $\overline{PSEN}$)		0.45	V	IOL = 3.2 mA (Notes 1, 2)
VOH	Output High Voltage (Ports 1, 2, 3, ALE/$\overline{PROG}$ and $\overline{PSEN}$)	2.4		V	IOH = −80 μA
VOH1	Output High Voltage (Port 0 in External Bus Mode)	2.4		V	IOH = −400 μA
IIL	Logical 0 Input Current (Ports 1, 2, 3 and RST)		−1	mA	Vin = 0.45 V
IIL1	Logical 0 Input Current ($\overline{EA}$)		−10	mA	Vin = VSS
IIL2	Logical 0 Input Current (XTAL2)		−3.2	mA	Vin = 0.45 V XTAL1 = VSS
ILI	Input Leakage Current (Port 0)		±10	μA	0.45 < Vin <VCC
IIH	Logical 1 Input Current ($\overline{EA}$)		1	mA	4.5V < Vin < 5.5V
IIH1	Input Current to RST to Activate Reset		500	μA	Vin < (VCC−1.5V)
ICC	Power Supply Current		175	mA	All Outputs Disconnected
CIO	Pin Capacitance		10	pF	Test Freq = 1MHz

ADVANCE INFORMATION
SEE INTEL FOR DESIGN-IN INFORMATION

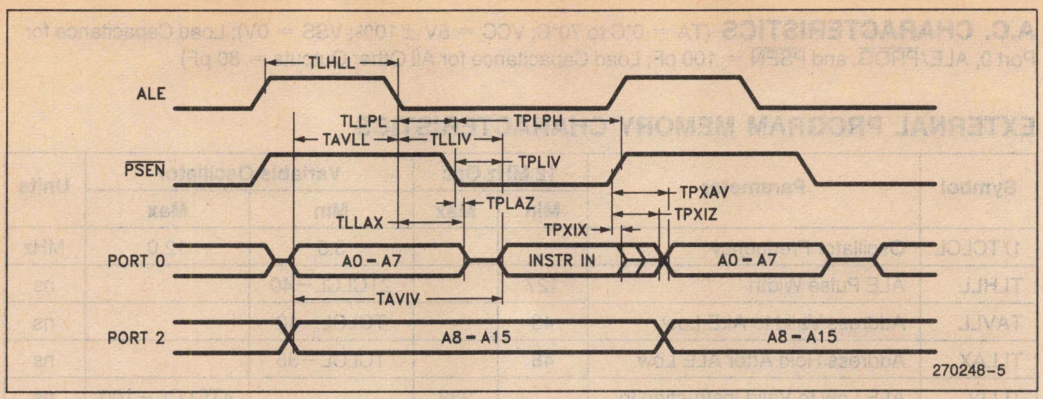
NOTES:
1. Capacitive loading on Ports 0 and 2 may cause spurious noise pulses to be superimposed on the VOLs of ALE/$\overline{PROG}$ and Ports 1 and 3. The noise is due to external bus capacitance discharging into the Port 0 and Port 2 pins when these pins make 1-to-0 transitions during bus operations. In the worst cases (capacitive loading > 100pF), the noise pulse on the ALE/$\overline{PROG}$ pin may exceed 0.8V. In such cases it may be desirable to qualify ALE with a Schmitt Trigger, or use an address latch with a Schmitt Trigger STROBE input.
2. ALE/$\overline{PROG}$ refers to a pin on the 8751BH. ALE refers to a timing signal that is output on the ALE/$\overline{PROG}$ pin.

A.C. CHARACTERISTICS (TA = 0°C to 70°C; VCC = 5V ±10%; VSS = 0V); Load Capacitance for Port 0, ALE/$\overline{PROG}$, and $\overline{PSEN}$ = 100 pF; Load Capacitance for All Other Outputs = 80 pF)

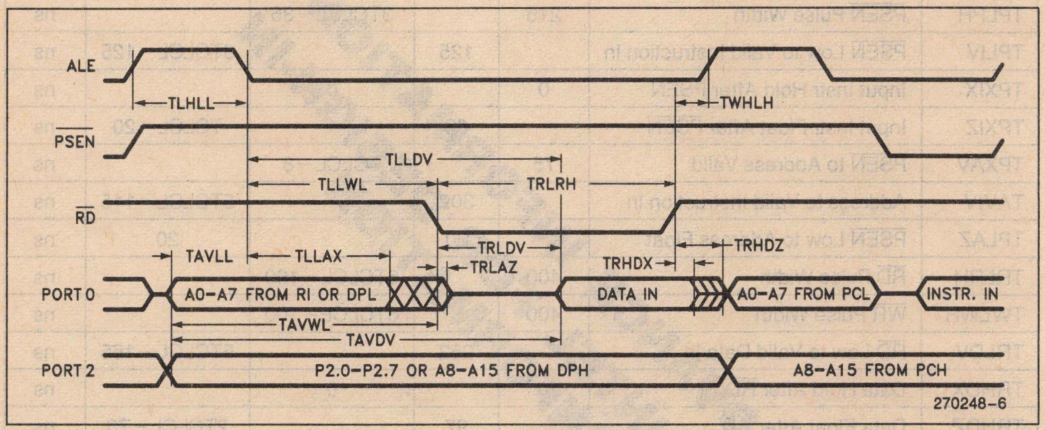
EXTERNAL PROGRAM MEMORY CHARACTERISTICS

Symbol	Parameter	12 MHz Osc		Variable Oscillator		Units
		Min	Max	Min	Max	
1/TCLCL	Oscillator Frequency			3.5	12.0	MHz
TLHLL	ALE Pulse Width	127		2TCLCL−40		ns
TAVLL	Address Valid to ALE Low	43		TCLCL−40		ns
TLLAX	Address Hold After ALE Low	48		TCLCL−35		ns
TLLIV	ALE Low to Valid Instruction In		233		4TCLCL−100	ns
TLLPL	ALE Low to $\overline{PSEN}$ Low	58		TCLCL−25		ns
TPLPH	$\overline{PSEN}$ Pulse Width	215		3TCLCL−35		ns
TPLIV	$\overline{PSEN}$ Low to Valid Instruction In		125		3TCLCL−125	ns
TPXIX	Input Instr Hold After $\overline{PSEN}$	0		0		ns
TPXIZ	Input Instr Float After $\overline{PSEN}$		63		TCLCL−20	ns
TPXAV	$\overline{PSEN}$ to Address Valid	75		TCLCL−8		ns
TAVIV	Address to Valid Instruction In		302		5TCLCL−115	ns
TPLAZ	$\overline{PSEN}$ Low to Address Float		20		20	ns
TRLRH	$\overline{RD}$ Pulse Width	400		6TCLCL−100		ns
TWLWH	$\overline{WR}$ Pulse Width	400		6TCLCL−100		ns
TRLDV	$\overline{RD}$ Low to Valid Data In		252		5TCLCL−165	ns
TRHDX	Data Hold After $\overline{RD}$	0		0		ns
TRHDZ	Data Float After $\overline{RD}$		97		2TCLCL−70	ns
TLLDV	ALE Low to Valid Data In		517		8TCLCL−150	ns
TAVDV	Address to Valid Data In		585		9TCLCL−165	ns
TLLWL	ALE Low to $\overline{RD}$ or $\overline{WR}$ Low	200	300	3TCLCL−50	3TCLCL+50	ns
TAVWL	Address to $\overline{RD}$ or $\overline{WR}$ Low	203		4TCLCL−130		ns
TQVWX	Data Valid to $\overline{WR}$ Transition	23		TCLCL−60		ns
TQVWH	Data Valid to $\overline{WR}$ High	433		7TCLCL−150		ns
TWHQX	Data Held After $\overline{WR}$	33		TCLCL−50		ns
TRLAZ	$\overline{RD}$ Low to Address Float		0		0	ns
TWHLH	$\overline{RD}$ or $\overline{WR}$ High to ALE High	43	123	TCLCL−40	TCLCL+40	ns

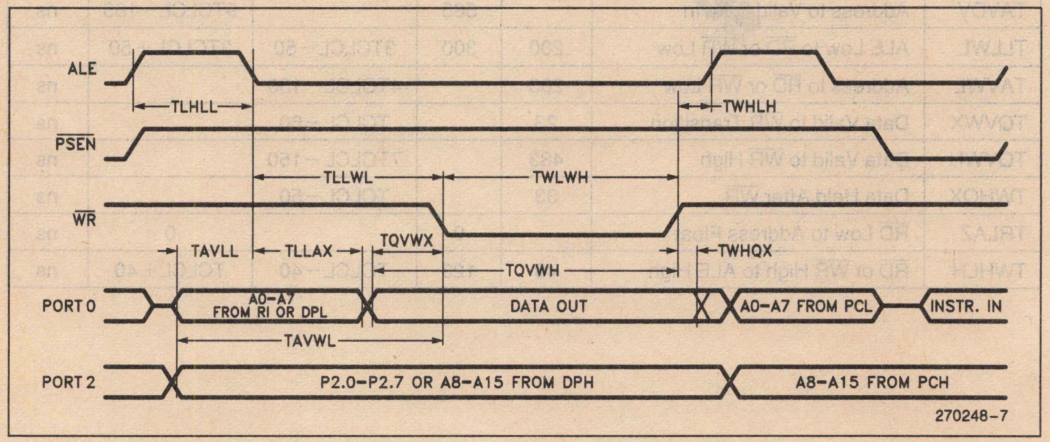
ADVANCE INFORMATION
SEE INTEL FOR DESIGN-IN INFORMATION



External Program Memory Read Cycle



External Data Memory Read Cycle

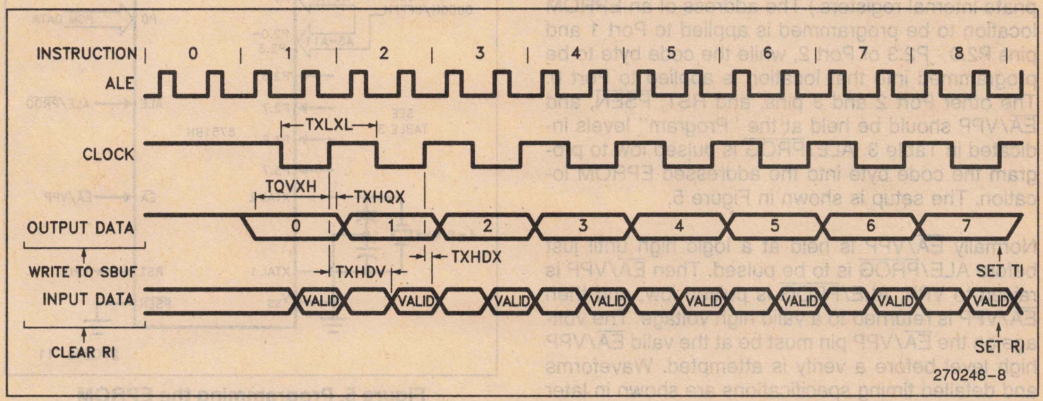


External Data Memory Write Cycle

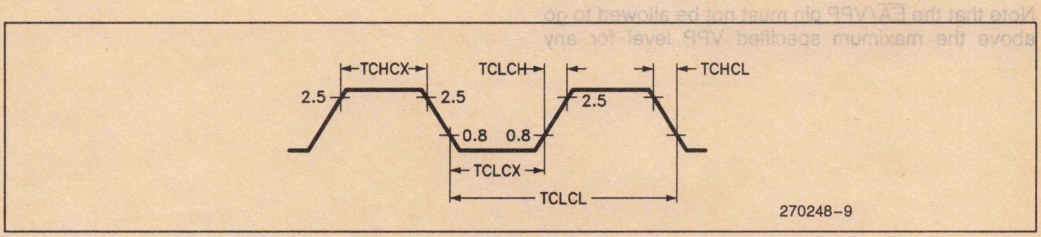
SERIAL PORT TIMING — SHIFT REGISTER MODE

TEST CONDITIONS (TA = 0°C to 70°C; VCC = 5V ± 10%; VSS = 0V; Load Capacitance = 80 pF)

Symbol	Parameter	12MHz Osc		Variable Oscillator		Units
		Min	Max	Min	Max	
TXLXL	Serial Port Clock Cycle Time	1.0		12TCLCL		μs
TQVXH	Output Data Setup to Clock Rising Edge	700		10TCLCL−133		ns
TXHQX	Output Data Hold After Clock Rising Edge	50		2TCLCL−117		ns
TXHDX	Input Data Hold After Clock Rising Edge	0		0		ns
TXHDV	Clock Rising Edge to Input Data Valid		700		10TCLCL−133	ns



Shift Register Mode Timing Waveforms

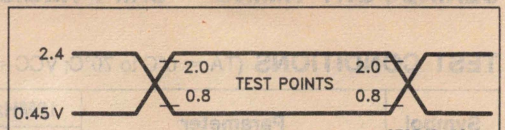


External Clock Drive Waveforms

EXTERNAL CLOCK DRIVE

Symbol	Parameter	Min	Max	Units
1/TCLCL	Oscillator Frequency	3.5	12	MHz
TCHCX	High Time	20		ns
TCLCX	Low Time	20		ns
TCLCH	Rise Time		20	ns
TCHCL	Fall Time		20	ns

AC TESTING INPUT/OUTPUT WAVEFORMS



AC inputs during testing are driven at 2.4V for a logic "1" and 0.45V for a logic "0". Timing measurements are made at 2.0V for a logic "1" and 0.8V for a logic "0".

EPROM CHARACTERISTICS

Programming the EPROM

To be programmed, the part must be running with a 4 to 6 MHz oscillator. (The reason the oscillator needs to be running is that the internal bus is being used to transfer address and program data to appropriate internal registers.) The address of an EPROM location to be programmed is applied to Port 1 and pins P2.0 - P2.3 of Port 2, while the code byte to be programmed into that location is applied to Port 0. The other Port 2 and 3 pins, and RST, $\overline{PSEN}$, and $\overline{EA}$/VPP should be held at the "Program" levels indicated in Table 3. ALE/$\overline{PROG}$ is pulsed low to program the code byte into the addressed EPROM location. The setup is shown in Figure 5.

Normally $\overline{EA}$/VPP is held at a logic high until just before ALE/$\overline{PROG}$ is to be pulsed. Then $\overline{EA}$/VPP is raised to VPP, ALE/$\overline{PROG}$ is pulsed low, and then $\overline{EA}$/VPP is returned to a valid high voltage. The voltage on the $\overline{EA}$/VPP pin must be at the valid $\overline{EA}$/VPP high level before a verify is attempted. Waveforms and detailed timing specifications are shown in later sections of this data sheet.

Note that the $\overline{EA}$/VPP pin must not be allowed to go above the maximum specified VPP level for any amount of time. Even a narrow glitch above that voltage level can cause permanent damage to the device. The VPP source should be well regulated and free of glitches.

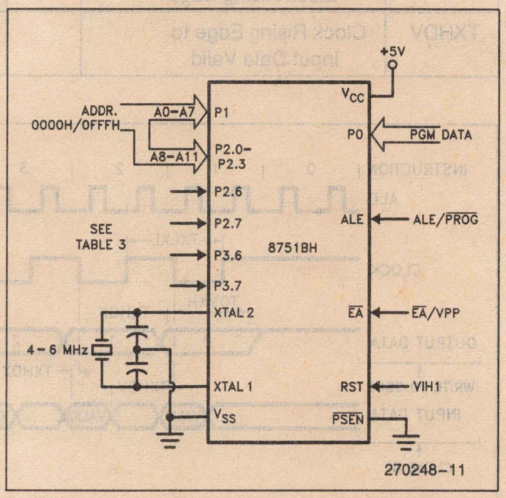


Figure 5. Programming the EPROM

Table 3. EPROM Programming Modes

MODE	RST	PSEN	ALE/PROG	EA/VPP	P2.7	P2.6	P3.6	P3.7
Program Code Data	1	0	0*	VPP	1	0	1	1
Verify Code Data	1	0	1	1	0	0	1	1
Program Encryption Table Use Addresses 0-1FH	1	0	0*	VPP	1	0	0	1
Program Lock x=1	1	0	0*	VPP	1	1	1	1
Bits (LBx) x=2	1	0	0*	VPP	1	1	0	0

NOTES:
"1" = Valid high for that pin
"0" = Valid low for that pin
"VPP" = +12.75V ±0.25V
* ALE/PROG is pulsed low for 100 uS for programming. (Quick-Pulse Programming™)

QUICK-PULSE PROGRAMMING™ ALGORITHM

The 8751BH can be programmed using the Quick-Pulse Programming Algorithm for microcontrollers. The features of the new programming method are a lower VPP (12.75 volts as compared to 21 volts) and a shorter programming pulse. It is possible to program the entire 4K Bytes of EPROM memory in less than 13 seconds with this algorithm!

To program the part using the new algorithm, VPP must be 12.75 ±0.25 Volts. ALE/PROG is pulsed low for 100 μseconds, 25 times. Then, the byte just programmed may be verified. After programming, the entire array should be verified. The Program Lock features are programmed using the same method, but with the setup as shown in Table 3. The only difference in programming Lock features is that the Lock features cannot be directly verified. Instead, verification of programming is by observing that their features are enabled.

PROGRAM VERIFICATION

If the Lock Bits have not been programmed, the on-chip Program Memory can be read out for verification purposes, if desired, either during or after the programming operation. The address of the Program Memory location to be read is applied to Port 1 and pins P2.0 - P2.3. The other pins should be held at the "Verify" levels indicated in Table 3. The contents of the addressed location will come out on Port 0. External pullups are required on Port 0 for this operation. (If the Encryption Array in the EPROM has been programmed, the data present at Port 0 will be Code Data XNOR Encryption Data. The user must know the Encryption Array contents to manually "unencrypt" the data during verify.)

The setup, which is shown in Figure 6, is the same as for programming the EPROM except that pin P2.7 is held at a logic low, or may be used as an active low read strobe.

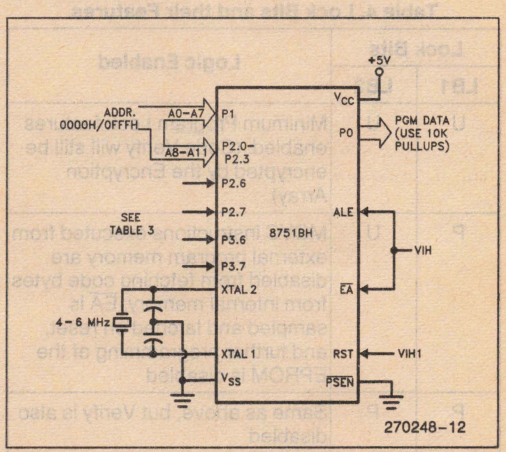


Figure 6. Verifying the EPROM

PROGRAM MEMORY LOCK

The two-level Program Lock system consists of 2 Lock bits and a 32-byte Encryption Array which are used to protect the program memory against software piracy.

ENCRYPTION ARRAY

Within the EPROM array are 32 bytes of Encryption Array that are initially unprogrammed (all 1s). Every time that a byte is addressed during a verify, 5 address lines are used to select a byte of the Encryption Array. This byte is then exclusive-NORed (XNOR) with the code byte, creating an Encrypted Verify byte. The algorithm, with the array in the unprogrammed state (all 1s), will return the code in its original, unmodified form.

It is recommended that whenever the Encryption Array is used, at least one of the Lock Bits be programmed as well.

LOCK BITS

Also included in the EPROM Program Lock scheme are two Lock Bits which function as shown in Table 4.

Table 4. Lock Bits and their Features

Lock Bits		Logic Enabled
LB1	LB2	
U	U	Minimum Program Lock features enabled. (Code Verify will still be encrypted by the Encryption Array)
P	U	MOVC instructions executed from external program memory are disabled from fetching code bytes from internal memory, $\overline{EA}$ is sampled and latched on reset, and further programming of the EPROM is disabled
P	P	Same as above, but Verify is also disabled
U	P	Reserved for Future Definition

P = Programmed
U = Unprogrammed

Erasing the EPROM also erases the Encryption Array and the Lock Bits, returning the part to full unlocked functionality.

To ensure proper functionality of the chip, the internally latched value of the $\overline{EA}$ pin must agree with its external state.

ERASURE CHARACTERISTICS

Erasure of the EPROM begins to occur when the chip is exposed to light with wavelengths shorter than approximately 4,000 Angstroms. Since sunlight and fluorescent lighting have wavelengths in this range, exposure to these light sources over an extended time (about 1 week in sunlight, or 3 years in room-level fluorescent lighting) could cause inadvertent erasure. If an application subjects the device to this type of exposure, it is suggested that an opaque label be placed over the window.

The recommended erasure procedure is exposure to ultraviolet light (at 2537 Angstroms) to an integrated dose of at lease 15 W-sec/cm. Exposing the EPROM to an ultraviolet lamp of 12,000 μW/cm rating for 20 to 30 minutes, at a distance of about 1 inch, should be sufficient.

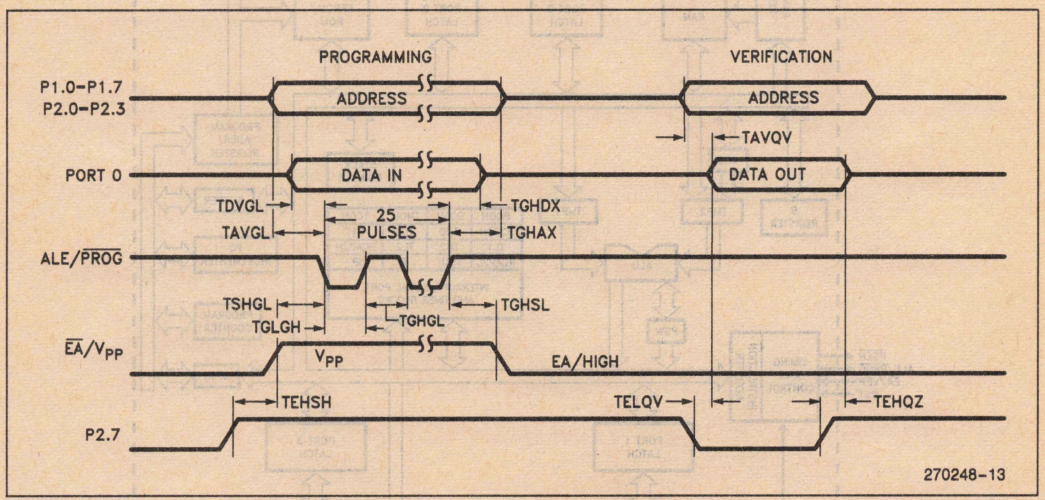
Erasure leaves the array in an all 1s state.

EPROM PROGRAMMING AND VERIFICATION CHARACTERISTICS

(TA = 21°C to 27°C, VCC = 5.0V ±10%, VSS = 0V)

Symbol	Parameter	Min	* Max	Units
VPP	Programming Supply Voltage	12.5	13.0	V
IPP	Programming Supply Current		50	mA
1/TCLCL	Oscillator Frequency	4	6	MHz
TAVGL	Address Setup to $\overline{PROG}$ Low	48TCLCL		
TGHAX	Address Hold After $\overline{PROG}$	48TCLCL		
TDVGL	Data Setup to $\overline{PROG}$ Low	48TCLCL		
TGHDX	Data Hold After $\overline{PROG}$	48TCLCL		
TEHSH	P2.7 ($\overline{ENABLE}$) High to VPP	48TCLCL		
TSHGL	VPP Setup to $\overline{PROG}$ Low	10		µsec
TGHSL	VPP Hold After $\overline{PROG}$	10		µsec
TGLGH	$\overline{PROG}$ Width	90	110	µsec
TAVQV	Address to Data Valid		48TCLCL	
TELQV	$\overline{ENABLE}$ Low to Data Valid		48TCLCL	
TEHQZ	Data Float After $\overline{ENABLE}$	0	48TCLCL	
TGHGL	$\overline{PROG}$ High to $\overline{PROG}$ Low	10		µsec

ADVANCE INFORMATION
SEE INTEL FOR DESIGN-IN
INFORMATION



EPROM Programming and Verification Waveforms

8032BH/8052BH/8752BH
SINGLE-CHIP 8-BIT MICROCOMPUTER
WITH 8K BYTES OF EPROM PROGRAM MEMORY

8032BH—ROMless
8052BH—8K Bytes of Factory Mask-Programmed ROM
8752BH—8K Bytes of EPROM Program Memory

- Program Memory Lock
- 256 Bytes Data Ram
- Quick Pulse Programming™ Algorithm
- 12.75 Volt Programming Voltage
- Boolean Processor
- 32 Programmable I/O Lines
- Three 16-Bit Timer/Counters
- 6 Interrupt Sources
- Programmable Serial Channel
- Separate Transmit/Receive Baud Rate Capability
- 64K External Program Memory Space
- 64K External Data Memory Space

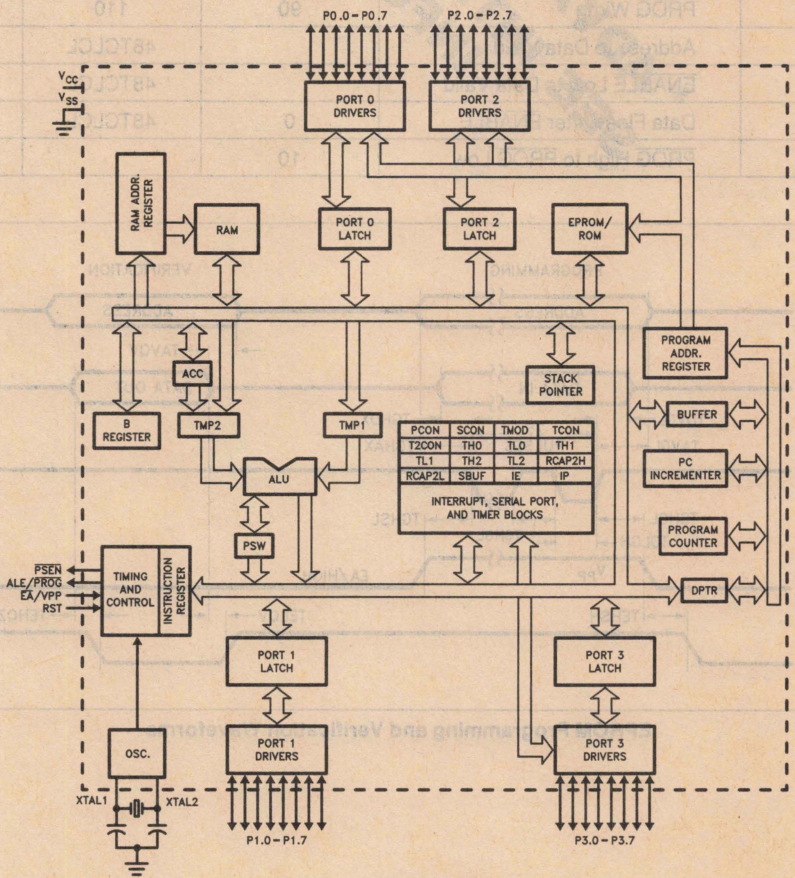


Figure 1. Block Diagram

270192-1

November 1986
Order Number: 270192-002

PIN DESCRIPTIONS

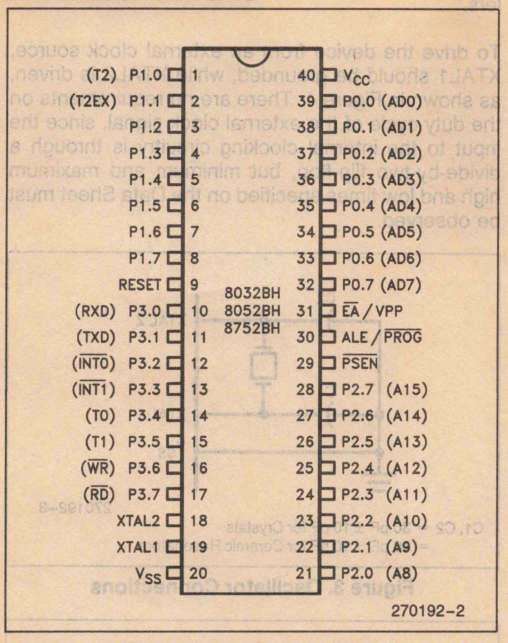


Figure 2. Pin Connections

VCC: Supply voltage.

VSS: Circuit ground.

Port 0: Port 0 is an 8-bit open drain bidirectional I/O port. As an output port each pin can sink 8 LS TTL inputs. Port 0 pins that have 1s written to them float, and in that state can be used as high-impedance inputs.

Port 0 is also the multiplexed low-order address and data bus during accesses to external Program and Data Memory. In this application it uses strong internal pullups when emitting 1s, and can source and sink 8 LS TTL inputs.

Port 0 also receives the code bytes during EPROM programming, and outputs the code bytes during program verification. External pullups are required during program verification.

Port 1: Port 1 is an 8-bit bidirectional I/O port with internal pullups. The Port 1 output buffers can sink/source 4 LS TTL inputs. Port 1 pins that have 1s written to them are pulled high by the internal pullups, and in that state can be used as inputs. As inputs, Port 1 pins that are externally being pulled low will source current (IIL, on the data sheet) because of the internal pullups.

Port 1 also receives the low-order address bytes during EPROM programming and program verification.

In addition, P1.0 and P1.1 serve the functions of the following special features of the MCS®-51 Family:

Port Pin	Alternate Function
P1.0	T2 (Timer/Counter 2 External Input)
P1.1	T2EX (Timer/Counter 2 Capture/Reload Trigger)

Port 2: Port 2 is an 8-bit bidirectional I/O port with internal pullups. The Port 2 output buffers can sink/source 4 LS TTL inputs. Port 2 pins that have 1s written to them are pulled high by the internal pullups, and in that state can be used as inputs. As inputs, Port 2 pins that are externally being pulled low will source current (IIL, on the data sheet) because of the internal pullups.

Port 2 emits the high-order address byte during fetches from external Program Memory and during accesses to external Data Memory that use 16-bit addresses (MOVX @DPTR). In this application it uses strong internal pullups when emitting 1s. During accesses to external Data Memory that use 8-bit addresses (MOVX @Ri), Port 2 emits the contents of the P2 Special Function Register.

Port 2 also receives the high-order address bits during EPROM programming and program verification.

Port 3: Port 3 is an 8-bit bidirectional I/O port with internal pullups. The Port 3 output buffers can sink/source 4 LS TTL inputs. Port 3 pins that have 1s written to them are pulled high by the internal pullups, and in that state can be used as inputs. As inputs, Port 3 pins that are externally being pulled low will source current (IIL, on the data sheet) because of the pullups.

Port 3 also serves the functions of various special features of the MCS®-51 Family, as listed below:

Port Pin	Alternate Function
P3.0	RXD (serial input port)
P3.1	TXD (serial output port)
P3.2	$\overline{INT0}$ (external interrupt 0)
P3.3	$\overline{INT1}$ (external interrupt 1)
P3.4	T0 (Timer 0 external input)
P3.5	T1 (Timer 1 external input)
P3.6	$\overline{WR}$ (external data memory write strobe)
P3.7	$\overline{RD}$ (external data memory read strobe)

RST: Reset input. A high on this pin for two machine cycles while the oscillator is running resets the device.

ALE/$\overline{PROG}$: Address Latch Enable output pulse for latching the low byte of the address during accesses to external memory. This pin is also the program pulse input (PROG) during EPROM programming on the 8752BH.

In normal operation ALE is emitted at a constant rate of 1/6 the oscillator frequency, and may be used for external timing or clocking purposes. Note, however, that one ALE pulse is skipped during each access to external Data Memory.

$\overline{PSEN}$: Program Store Enable is the Read strobe to External Program Memory.

When the device is executing code from external Program Memory, $\overline{PSEN}$ is activated twice each machine cycle, except that two $\overline{PSEN}$ activations are skipped during each access to External Data Memory.

$\overline{EA}$/VPP: External Access enable. $\overline{EA}$ must be strapped low in order to enable the device to fetch code from External Program Memory locations 0000H to 1FFFH. Note, however, that if either of the Lock Bits are programmed, $\overline{EA}$ will be internally latched on reset.

$\overline{EA}$ should be strapped to VCC for internal program executions.

This pin also receives the 12.75V programming supply voltage (VPP) during EPROM programming.

XTAL1: Input to the inverting oscillator amplifier.

XTAL2: Output from the inverting oscillator amplifier.

OSCILLATOR CHARACTERISTICS

XTAL1 and XTAL2 are the input and output, respectively, of an inverting amplifier which can be configured for use as an on-chip oscillator, as shown in Figure 3. Either a quartz crystal or ceramic resonator may be used. More detailed information concerning the use of the on-chip oscillator is available in Application Note AP-155, "Oscillators for Microcontrollers."

To drive the device from an external clock source, XTAL1 should be grounded, while XTAL2 is driven, as shown in Figure 4. There are no requirements on the duty cycle of the external clock signal, since the input to the internal clocking circuitry is through a divide-by-two flip-flop, but minimum and maximum high and low times specified on the Data Sheet must be observed.

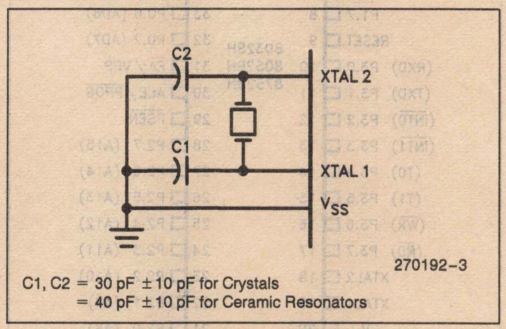


Figure 3. Oscillator Connections

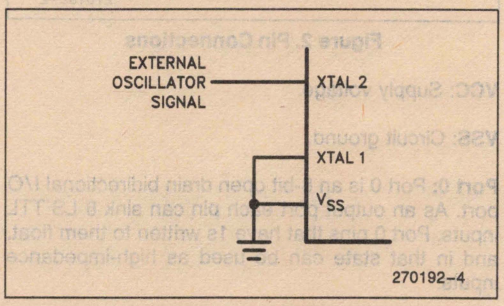


Figure 4. External Clock Drive Configuration

DESIGN CONSIDERATIONS

Exposure to light when the 8752BH is in operation may cause logic errors. For this reason, it is suggested that an opaque label be placed over the window of the 8752BH when the die is exposed to ambient light.

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias .....0°C to 70°C
Storage Temperature .....−65°C to +150°C
Voltage on $\overline{EA}$/VPP Pin to VSS ..−0.5V to +13.0V
Voltage on Any Other Pin to VSS....−0.5V to +7V
Power Dissipation.....1.5W
(based on PACKAGE heat transfer limitations, not device power consumption)

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

NOTICE: Specifications contained within the following tables are subject to change.

D.C. CHARACTERISTICS (TA = 0°C to 70°C; VCC = 5V ±10%; VSS = 0V)

Symbol	Parameter	Min	Max	Unit	Test Conditions
VIL	Input Low Voltage (Except $\overline{EA}$)	−0.5	0.8	V	
VIL1	Input Low Voltage $\overline{EA}$	VSS	0.7	V	
VIH	Input High Voltage (Except XTAL2, RST, $\overline{EA}$)	2.0	VCC+0.5	V	
VIH1	Input High Voltage XTAL2, RST	2.5	VCC+0.5	V	XTAL1 = VSS
VIH2	Input High Voltage to $\overline{EA}$	4.5	5.5	V	
VOL	Output Low Voltage (Ports 1, 2 and 3)		0.45	V	IOL = 1.6 mA (Note 1)
VOL1	Output Low Voltage (Port 0, ALE/$\overline{PROG}$, $\overline{PSEN}$)		0.45	V	IOL = 3.2 mA (Note 1, 2)
VOH	Output High Voltage (Ports 1, 2, 3, ALE/$\overline{PROG}$ and $\overline{PSEN}$)	2.4		V	IOH = −80 μA
VOH1	Output High Voltage (Port 0 in External Bus Mode)	2.4		V	IOH = −400 μA
IIL	Logical 0 Input Current (Ports 1, 2, 3 and RST)		−500	μA	Vin = 0.45 V
IIL1	Logical 0 Input Current ($\overline{EA}$)	−10	500	mA μA	Vin = VSS
IIL2	Logical 0 Input Current (XTAL2)		−3.2	mA	Vin = 0.45V XTAL1 = VSS
ILI	Input Leakage Current (Port 0)		±10	μA	0.45 < Vin <VCC
IIH	Logical 1 Input Current ($\overline{EA}$)		1	mA	4.5V < Vin < 5.5V
IIH1	Input Current to RST to activate Reset		500	μA	Vin < (VCC−1.5V)
ICC	Power Supply Current		175	mA	All Outputs Disconnected
CIO	Pin Capacitance		10	pF	Test freq = 1MHz

ADVANCE INFORMATION
SEE INTEL FOR DESIGN-IN INFORMATION

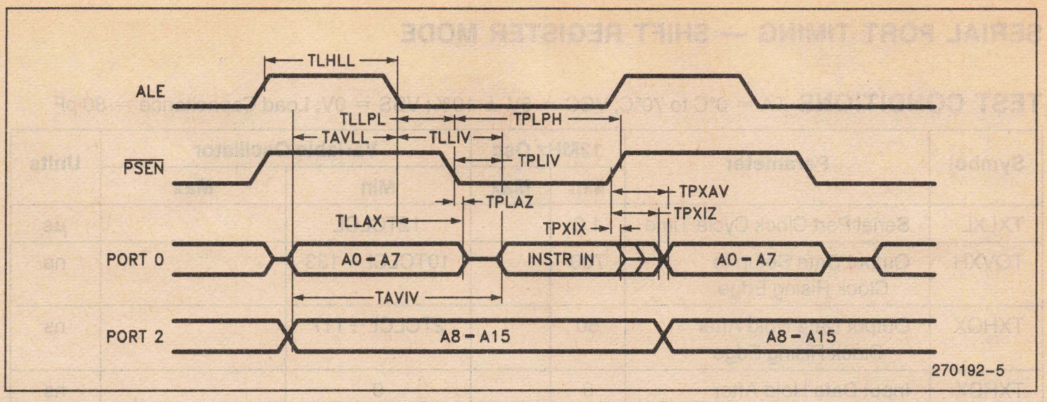
NOTES:
1. Capacitive loading on Ports 0 and 2 may cause spurious noise pulses to be superimposed on the VOLs of ALE/$\overline{PROG}$ and Ports 1 and 3. The noise is due to external bus capacitance discharging into the Port 0 and Port 2 pins when these pins make 1-to-0 transitions during bus operations. In the worst cases (capacitive loading > 100 pF), the noise pulse on the ALE/$\overline{PROG}$ pin may exceed 0.8V. In such cases it may be desirable to qualify ALE with a Schmitt Trigger, or use an address latch with a Schmitt Trigger STROBE input.
2. ALE/$\overline{PROG}$ refers to a pin on the device. ALE refers to a timing signal that is output on the ALE/$\overline{PROG}$ pin.

A.C. CHARACTERISTICS (TA = 0°C to 70°C; VCC = 5V ±10%; VSS = 0V); Load Capacitance for Port 0, ALE/$\overline{PROG}$, and $\overline{PSEN}$ = 100 pF; Load Capacitance for All Other Outputs = 80 pF)

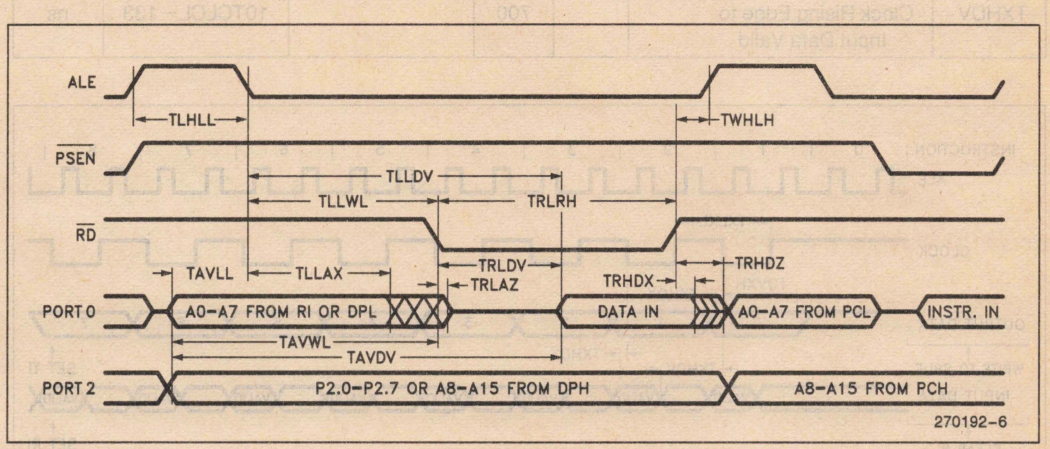
EXTERNAL PROGRAM MEMORY CHARACTERISTICS

Symbol	Parameter	12 MHz Osc		Variable Oscillator		Units
		Min	Max	Min	Max	
1/TCLCL	Oscillator Frequency			3.5	12.0	MHz
TLHLL	ALE Pulse Width	127		2TCLCL−40		ns
TAVLL	Address Valid to ALE Low	43		TCLCL−40		ns
TLLAX	Address Hold After ALE Low	48		TCLCL−35		ns
TLLIV	ALE Low to Valid Instruction In		233		4TCLCL−100	ns
TLLPL	ALE Low to $\overline{PSEN}$ Low	58		TCLCL−25		ns
TPLPH	$\overline{PSEN}$ Pulse Width	215		3TCLCL−35		ns
TPLIV	$\overline{PSEN}$ Low to Valid Instruction In		125		3TCLCL−125	ns
TPXIX	Input Instr Hold After $\overline{PSEN}$	0		0		ns
TPXIZ	Input Instr Float After $\overline{PSEN}$		63		TCLCL−20	ns
TPXAV	$\overline{PSEN}$ to Address Valid	75		TCLCL−8		ns
TAVIV	Address to Valid Instruction In		302		5TCLCL−115	ns
TPLAZ	$\overline{PSEN}$ Low to Address Float		20		20	ns
TRLRH	$\overline{RD}$ Pulse Width	400		6TCLCL−100		ns
TWLWH	$\overline{WR}$ Pulse Width	400		6TCLCL−100		ns
TRLDV	$\overline{RD}$ Low to Valid Data In		252		5TCLCL−165	ns
TRHDX	Data Hold After $\overline{RD}$	0		0		ns
TRHDZ	Data Float After $\overline{RD}$		97		2TCLCL−70	ns
TLLDV	ALE Low to Valid Data In		517		8TCLCL−150	ns
TAVDV	Address to Valid Data In		585		9TCLCL−165	ns
TLLWL	ALE Low to $\overline{RD}$ or $\overline{WR}$ Low	200	300	3TCLCL−50	3TCLCL+50	ns
TAVWL	Address to $\overline{RD}$ or $\overline{WR}$ Low	203		4TCLCL−130		ns
TQVWX	Data Valid to $\overline{WR}$ Transition	23		TCLCL−60		ns
TQVWH	Data Valid to $\overline{WR}$ High	433		7TCLCL−150		ns
TWHQX	Data Held After $\overline{WR}$	33		TCLCL−50		ns
TRLAZ	$\overline{RD}$ Low to Address Float		0		0	ns
TWHLH	$\overline{RD}$ or $\overline{WR}$ High to ALE High	43	123	TCLCL−40	TCLCL+40	ns

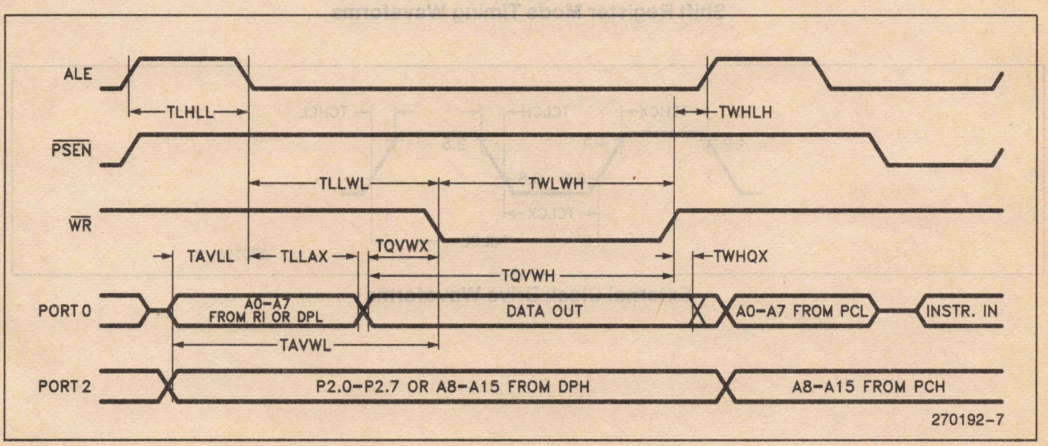
ADVANCE INFORMATION
SEE INTEL FOR DESIGN-IN INFORMATION



External Program Memory Read Cycle



External Data Memory Read Cycle

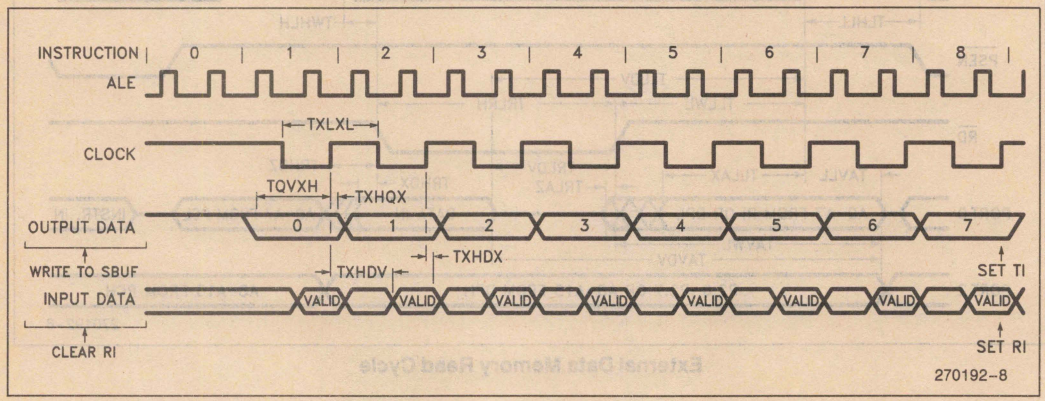


External Data Memory Write Cycle

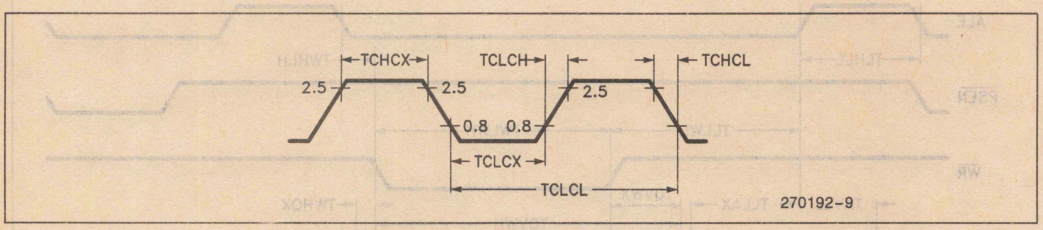
SERIAL PORT TIMING — SHIFT REGISTER MODE

TEST CONDITIONS TA = 0°C to 70°C; VCC = 5V ± 10%; VSS = 0V; Load Capacitance = 80 pF

Symbol	Parameter	12MHz Osc		Variable Oscillator		Units
		Min	Max	Min	Max	
TXLXL	Serial Port Clock Cycle Time	1.0		12TCLCL		μs
TQVXH	Output Data Setup to Clock Rising Edge	700		10TCLCL−133		ns
TXHQX	Output Data Hold After Clock Rising Edge	50		2TCLCL−117		ns
TXHDX	Input Data Hold After Clock Rising Edge	0		0		ns
TXHDV	Clock Rising Edge to Input Data Valid		700		10TCLCL−133	ns



Shift Register Mode Timing Waveforms

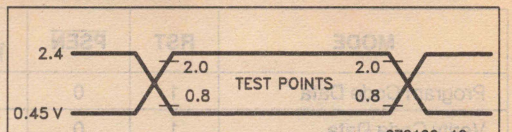


External Clock Drive Waveforms

EXTERNAL CLOCK DRIVE

Symbol	Parameter	Min	Max	Units
1/TCLCL	Oscillator Frequency	3.5	12	MHz
TCHCX	High Time	20		ns
TCLCX	Low Time	20		ns
TCLCH	Rise Time		20	ns
TCHCL	Fall Time		20	ns

AC TESTING INPUT/OUTPUT WAVEFORMS



AC inputs during testing are driven at 2.4V for a logic "1" and 0.45V for a logic "0". Timing measurements are made at 2.0V for a logic "1" and 0.8V for a logic "0".

EPROM CHARACTERISTICS

Programming the EPROM

To be programmed, the 8752BH must be running with a 4 to 6 MHz oscillator. (The reason the oscillator needs to be running is that the internal bus is being used to transfer address and program data to appropriate internal registers.) The address of an EPROM location to be programmed is applied to Port 1 and pins P2.0 - P2.4 of Port 2, while the code byte to be programmed into that location is applied to Port 0. The other Port 2 and 3 pins, and RST, $\overline{PSEN}$, and $\overline{EA}$/VPP should be held at the "Program" levels indicated in Table 3. ALE/$\overline{PROG}$ is pulsed low to program the code byte into the addressed EPROM location. The setup is shown in Figure 5.

Normally $\overline{EA}$/VPP is held at a logic high until just before ALE/$\overline{PROG}$ is to be pulsed. Then $\overline{EA}$/VPP is raised to VPP, ALE/$\overline{PROG}$ is pulsed low, and then $\overline{EA}$/VPP is returned to a valid high voltage. The voltage on the $\overline{EA}$/VPP pin must be at the valid $\overline{EA}$/VPP high level before a verify is attempted. Waveforms and detailed timing specifications are shown in later sections of this data sheet.

Note that the $\overline{EA}$/VPP pin must not be allowed to go above the maximum specified VPP level for any amount of time. Even a narrow glitch above that voltage level can cause permanent damage to the device. The VPP source should be well regulated and free of glitches.

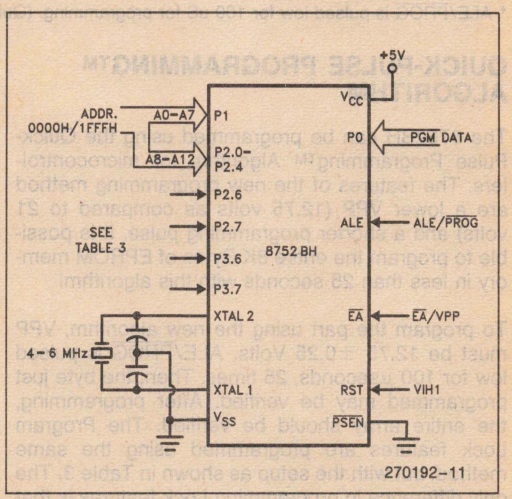


Figure 5. Programming the EPROM

Table 3. EPROM Programming Modes

MODE	RST	$\overline{PSEN}$	ALE/$\overline{PROG}$	$\overline{EA}$/VPP	P2.7	P2.6	P3.6	P3.7
Program Code Data	1	0	0*	VPP	1	0	1	1
Verify Code Data	1	0	1	1	0	0	1	1
Program Encryption Table Use Addresses 0-1FH	1	0	0*	VPP	1	0	0	1
Program Lock x=1	1	0	0*	VPP	1	1	1	1
Bits (LBx) x=2	1	0	0*	VPP	1	1	0	0

NOTES:
"1" = Valid high for that pin
"0" = Valid low for that pin
"VPP" = +12.75V ±0.25V
* ALE/$\overline{PROG}$ is pulsed low for 100 uS for programming. (Quick-Pulse Programming™)

QUICK-PULSE PROGRAMMING™ ALGORITHM

The 8752BH can be programmed using the Quick-Pulse Programming™ Algorithm for microcontrollers. The features of the new programming method are a lower VPP (12.75 volts as compared to 21 volts) and a shorter programming pulse. It is possible to program the entire 8K Bytes of EPROM memory in less than 25 seconds with this algorithm!

To program the part using the new algorithm, VPP must be 12.75 ±0.25 Volts. ALE/$\overline{PROG}$ is pulsed low for 100 μseconds, 25 times. Then, the byte just programmed may be verified. After programming, the entire array should be verified. The Program Lock features are programmed using the same method, but with the setup as shown in Table 3. The only difference in programming Lock features is that the Lock features cannot be directly verified. Instead, verification of programming is by observing that their features are enabled.

PROGRAM VERIFICATION

If the Lock Bits have not been programmed, the on-chip Program Memory can be read out for verification purposes, if desired, either during or after the programming operation. The address of the Program Memory location to be read is applied to Port 1 and pins P2.0 - P2.4. The other pins should be held at the "Verify" levels indicated in Table 3. The contents of the addressed location will come out on Port 0. External pullups are required on Port 0 for this operation. (If the Encryption Array in the EPROM has been programmed, the data present at Port 0 will be Code Data XNOR Encryption Data. The user must know the Encryption Array contents to manually "unencrypt" the data during verify.)

The setup, which is shown in Figure 6, is the same as for programming the EPROM except that pin P2.7 is held at a logic low, or may be used as an active low read strobe.

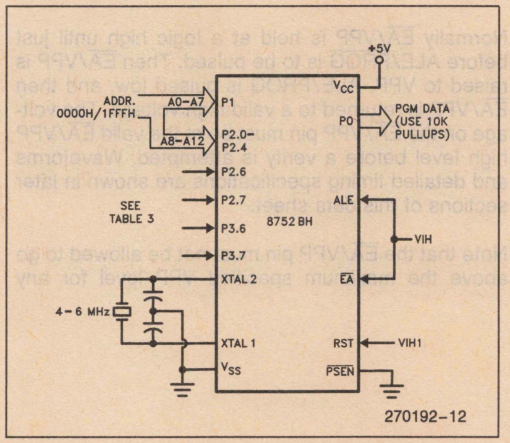


Figure 6. Verifying the EPROM

PROGRAM MEMORY LOCK

The two-level Program Lock system consists of 2 Lock bits and a 32-byte Encryption Array which are used to protect the program memory against software piracy. The following description applies to the 8752BH. The same options are also available on the 8052BH, mask-programmed at the factory.

ENCRYPTION ARRAY

Within the EPROM array are 32 bytes of Encryption Array that are initially unprogrammed (all 1s). Every time that a byte is addressed during a verify, 5 address lines are used to select a byte of the Encryption Array. This byte is then exclusive-NORed (XNOR) with the code byte, creating an Encrypted Verify byte. The algorithm, with the array in the unprogrammed state (all 1s), will return the code in its original, unmodified form.

It is recommended that whenever the Encryption Array is used, at least one of the Lock Bits be programmed as well.

LOCK BITS

Also included in the EPROM Program Lock scheme are two Lock Bits which function as shown in Table 4.

Table 4. Lock Bits and their Features

Lock Bits		Logic Enabled
LB1	LB2	
U	U	Minimum Program Lock features enabled. (Code Verify will still be encrypted by the Encryption Array)
P	U	MOVC instructions executed from external program memory are disabled from fetching code bytes from internal memory, $\overline{EA}$ is sampled and latched on reset, and further programming of the EPROM is disabled
P	P	Same as above, but Verify is also disabled
U	P	Reserved for Future Definition

P = Programmed
U = Unprogrammed

Erasing the EPROM also erases the Encryption Array and the Lock Bits, returning the part to full unlocked functionality.

To ensure proper functionality of the chip, the internally latched value of the $\overline{EA}$ pin must agree with its external state.

ERASURE CHARACTERISTICS

Erasure of the EPROM begins to occur when the 8752BH is exposed to light with wavelengths shorter than approximately 4,000 Angstroms. Since sunlight and fluorescent lighting have wavelengths in this range, exposure to these light sources over an extended time (about 1 week in sunlight, or 3 years in room-level fluorescent lighting) could cause inadvertent erasure. If an application subjects the device to this type of exposure, it is suggested that an opaque label be placed over the window.

The recommended erasure procedure is exposure to ultraviolet light (at 2537 Angstroms) to an integrated dose of at lease 15 W-sec/cm. Exposing the EPROM to an ultraviolet lamp of 12,000 μW/cm rating for 20 to 30 minutes, at a distance of about 1 inch, should be sufficient.

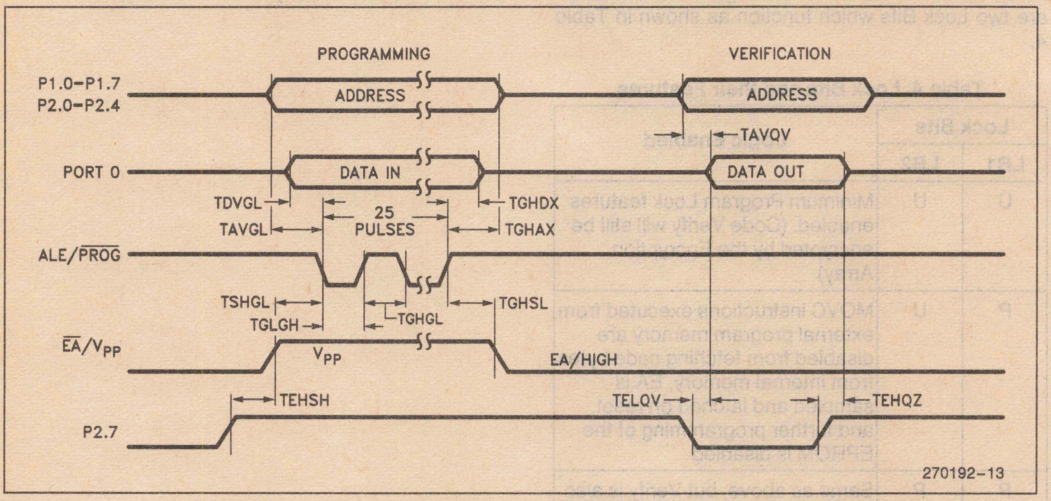
Erasure leaves the array in an all 1s state.

EPROM PROGRAMMING AND VERIFICATION CHARACTERISTICS

(TA = 21°C to 27°C, VCC = 5.0V ±10%, VSS = 0V)

Symbol	Parameter	Min	* Max	Units
VPP	Programming Supply Voltage	12.5	13.0	V
IPP	Programming Supply Current		50	mA
1/TCLCL	Oscillator Frequency	4	6	MHz
TAVGL	Address Setup to $\overline{PROG}$ Low	48TCLCL		
TGHAX	Address Hold After $\overline{PROG}$	48TCLCL		
TDVGL	Data Setup to $\overline{PROG}$ Low	48TCLCL		
TGHDX	Data Hold After $\overline{PROG}$	48TCLCL		
TEHSH	P2.7 ($\overline{ENABLE}$) High to VPP	48TCLCL		
TSHGL	VPP Setup to $\overline{PROG}$ Low	10		µsec
TGHSL	VPP Hold After $\overline{PROG}$	10		µsec
TGLGH	$\overline{PROG}$ Width	90	110	µsec
TAVQV	Address to Data Valid		48TCLCL	
TELQV	$\overline{ENABLE}$ Low to Data Valid		48TCLCL	
TEHQZ	Data Float After $\overline{ENABLE}$	0	48TCLCL	
TGHGL	$\overline{PROG}$ High to $\overline{PROG}$ Low	10		µsec

ADVANCE INFORMATION
SEE INTEL FOR DESIGN-IN INFORMATION



EPROM Programming and Verification Waveforms

80C51BH/80C51BH-1/80C51BH-2 CHMOS SINGLE-CHIP 8-BIT MICROCOMPUTER WITH FACTORY MASK-PROGRAMMABLE ROM

80C31BH/80C31BH-1/80C31BH-2 CHMOS SINGLE-CHIP 8-BIT CONTROL-ORIENTED CPU WITH RAM AND I/O

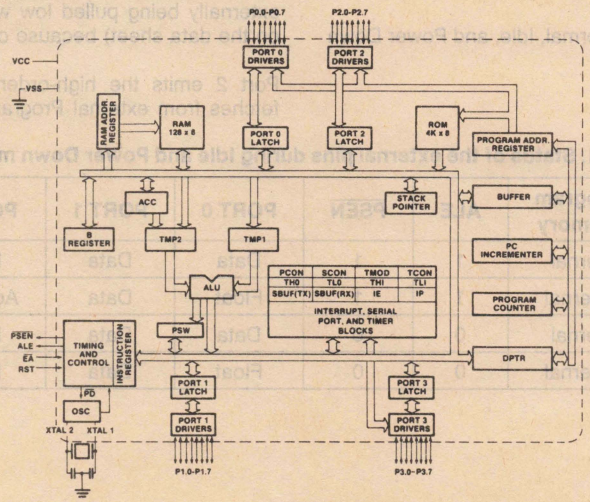
80C51BH/80C31BH—3.5 to 12 MHz, V_{CC} = 5V ±20%
80C51BH-1/80C31BH-1—3.5 to 16 MHz, V_{CC} = 5V ±20%
80C51BH-2/80C31BH-2—0.5 to 12 MHz, V_{CC} = 5V ±20%

- **Power Control Modes**
- **128 x 8-Bit RAM**
- **32 Programmable I/O Lines**
- **Two 16-Bit Timer/Counters**
- **64K Program Memory Space**
- **High Performance CHMOS Process**
- **Boolean Processor**
- **5 Interrupt Sources**
- **Programmable Serial Port**
- **64K Data Memory Space**

The MCS®-51 CHMOS products are fabricated on Intel's CHMOS III process and are functionally compatible with the standard MCS-51 HMOS and EPROM products. CHMOS III is a technology which combines the high speed and density characteristics of HMOS with the low power attributes of CHMOS. This combination expands the effectiveness of the powerful MCS-51 architecture and instruction set.

Like the MCS-51 HMOS versions, the MCS-51 CHMOS products have the following features: 4K byte of ROM (80C51BH/80C51BH-1/80C51BH-2 only); 128 bytes of RAM; 32 I/O lines; two 16-bit timer/counters; a five-source two-level interrupt structure; a full duplex serial port; and on-chip oscillator and clock circuitry. In addition, the MCS-51 CHMOS products have two software selectable modes of reduced activity for further power reduction—Idle and Power Down.

The Idle mode freezes the CPU while allowing the RAM, timer/counters serial port and interrupt system to continue functioning. The Power Down mode saves the RAM contents but freezes the oscillator, causing all other chip functions to be inoperative.



270064-1

Figure 1. Block Diagram

November 1986
Order Number: 270064-004

IDLE MODE

In the Idle mode, the CPU puts itself to sleep while all the on chip peripherals stay active. The instruction that invokes the Idle mode is the last instruction executed in the normal operating mode before Idle mode is activated. The content of CPU, the on chip RAM, and all the Special Function Registers remain intact during this mode. The Idle mode can be terminated either by any enabled interrupt, at which time the process is picked up at the interrupt service routine and continued, or by a hardware reset which starts the processor the same as a power on reset.

POWER DOWN MODE

In the Power Down mode the oscillator is stopped, and the instruction that invokes Power Down is the last instruction executed. Only the contents of the on chip RAM is preserved. A hardware reset is the only way to terminate Power Down.

The control bits for the reduced power modes are in the Special Function Register PCON.

NOTE:
For more detailed information on these reduced power modes refer to Application Note AP-252, "Designing with the 80C51BH".

PIN DESCRIPTIONS

V_{CC}

Supply voltage during normal, Idle, and Power Down operations.

V_{SS}

Circuit ground.

Port 0

Port 0 is an 8-bit open drain bi-directional I/O port. Port 0 pins that have 1's written to them float, and in that state can be used as high-impedance inputs.

Port 0 is also the multiplexed low-order address and data bus during accesses to external Program and Data Memory. In this application it uses strong internal pullups when emitting 1s. Port 0 also outputs the code bytes during program verification in the 80C51BH. External pullups are required during program verification.

Port 1

Port 1 is an 8-bit bidirectional I/O port with internal pullups. Port 1 pins that have 1s written to them are pulled high by the internal pullups, and in that state can be used as inputs. As inputs, Port 1 pins that are externally being pulled low will source current (IIL, on the data sheet) because of the internal pullups.

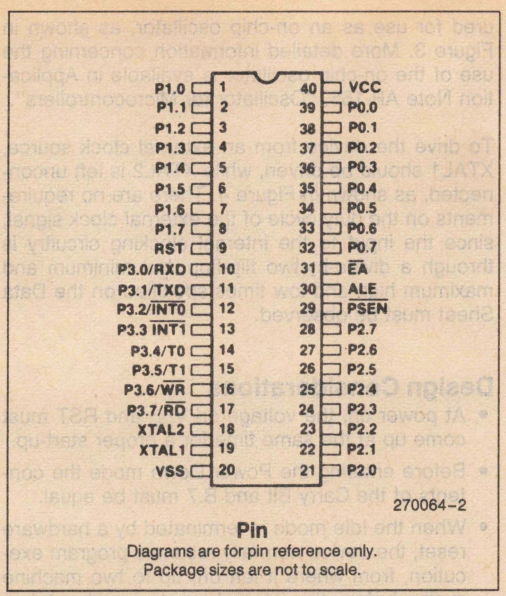
Port 2

Port 2 is an 8-bit bidirectional I/O port with internal pullups. Port 2 pins that have 1s written to them are pulled high by the internal pullups, and in that state can be used as inputs. As inputs, Port 2 pins that are externally being pulled low will source current (IIL, on the data sheet) because of the internal pullups.

Port 2 emits the high-order address byte during fetches from external Program Memory and during

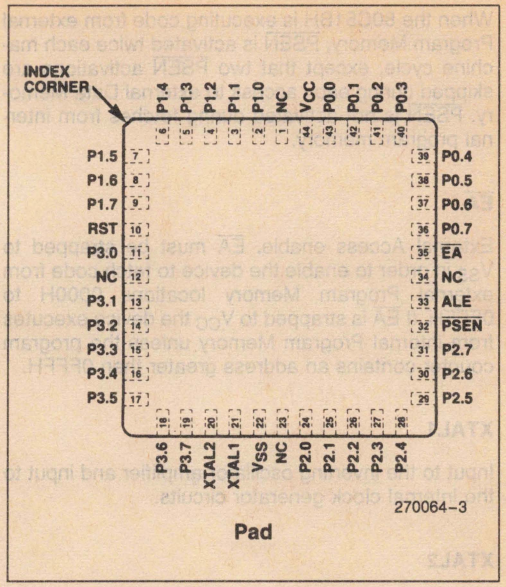
Table 1. Status of the external pins during Idle and Power Down modes

Mode	Program Memory	ALE	$\overline{PSEN}$	PORT 0	PORT 1	PORT 2	PORT 3
Idle	Internal	1	1	Data	Data	Data	Data
Idle	External	1	1	Float	Data	Address	Data
Power Down	Internal	0	0	Data	Data	Data	Data
Power Down	External	0	0	Float	Data	Data	Data



Pin

Diagrams are for pin reference only.
Package sizes are not to scale.



Pad

Figure 2. Connection Diagrams

accesses to external Data Memory that use 16-bit addresses (MOVX @DPTR). In this application it uses strong internal pullups when emitting 1s. During accesses to external Data Memory that use 8-bit addresses (MOVX @Ri), Port 2 emits the contents of the P2 Special Function Register.

Port 3

Port 3 is an 8-bit bidirectional I/O port with internal pullups. Port 3 pins that have 1s written to them are pulled high by the internal pullups, and in that state can be used as inputs. As inputs, Port 3 pins that are externally being pulled low will source current (IIL, on the data sheet) because of the pullups.

Port 3 also serves the functions of various special features of the MCS-51 Family, as listed below:

Port Pin	Alternate Function
P3.0	RXD (serial input port)
P3.1	TXD (serial output port)
P3.2	$\overline{INT0}$ (external interrupt 0)
P3.3	$\overline{INT1}$ (external interrupt 1)
P3.4	T0 (Timer 0 external input)
P3.5	T1 (Timer 1 external input)
P3.6	$\overline{WR}$ (external data memory write strobe)
P3.7	$\overline{RD}$ (external data memory read strobe)

RST

Reset input. A high on this pin for two machine cycles while the oscillator is running resets the device. An internal diffused resistor to V_{SS} permits Power-On reset using only an external capacitor to V_{CC}.

ALE

Address Latch Enable output pulse for latching the low byte of the address during accesses to external memory.

In normal operation ALE is emitted at a constant rate of 1/6 the oscillator frequency, and may be used for external timing or clocking purposes. Note, however, that one ALE pulse is skipped during each access to external Data Memory.

$\overline{PSEN}$

Program Store Enable is the read strobe to external Program Memory.

When the 80C51BH is executing code from external Program Memory, $\overline{PSEN}$ is activated twice each machine cycle, except that two $\overline{PSEN}$ activations are skipped during each access to external Data Memory. $\overline{PSEN}$ is not activated during fetches from internal program memory.

$\overline{EA}$

External Access enable. $\overline{EA}$ must be strapped to V_{SS} in order to enable the device to fetch code from external Program Memory locations 0000H to 0FFFH. If $\overline{EA}$ is strapped to V_{CC} the device executes from internal Program Memory unless the program counter contains an address greater than 0FFFH.

XTAL1

Input to the inverting oscillator amplifier and input to the internal clock generator circuits.

XTAL2

Output from the inverting oscillator amplifier.

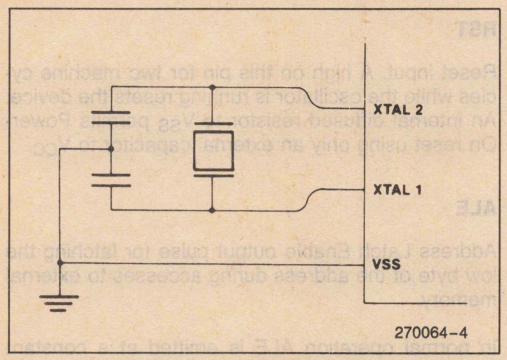


Figure 3. Crystal Oscillator

Oscillator Characteristics

XTAL1 and XTAL2 are the input and output, respectively, of an inverting amplifier which can be configured for use as an on-chip oscillator, as shown in Figure 3. More detailed information concerning the use of the on-chip oscillator is available in Application Note AP-155, "Oscillator for Microcontrollers".

To drive the device from an external clock source, XTAL1 should be driven, while XTAL2 is left unconnected, as shown in Figure 4. There are no requirements on the duty cycle of the external clock signal, since the input to the internal clocking circuitry is through a divide-by-two flip-flop, but minimum and maximum high and low times specified on the Data Sheet must be observed.

Design Considerations

- At power on, the voltage on V_{CC} and RST must come up at the same time for a proper start-up.
- Before entering the Power Down mode the contents of the Carry Bit and B.7 must be equal.
- When the Idle mode is terminated by a hardware reset, the device normally resumes program execution, from where it left off, up to two machine cycles before the internal reset algorithm takes control. On-chip hardware inhibits access to internal RAM in this event, but access to the port pins is not inhibited. To eliminate the possibility of an unexpected write when Idle is terminated by reset, the instruction following the one that invokes Idle should not be one that writes to a port pin or to external memory.

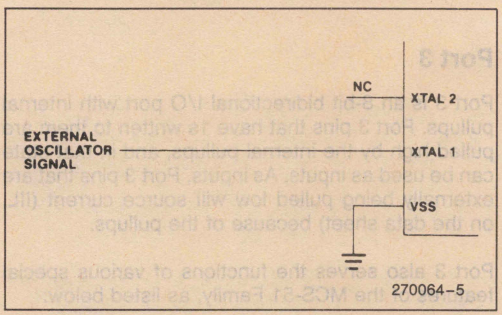


Figure 4. External Drive Configuration

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias .....0°C to 70°C

Storage Temperature .....−65°C to +150°C

Voltage on any Pin to V_{SS} .....−0.5V to V_{CC} +0.5V

Voltage on V_{CC} to V_{SS} .....−0.5V to 6.5V

Power Dissipation.....1.0W*

*This value is based on the maximum allowable die temperature and the thermal resistance of the package.

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

NOTICE: Specifications contained within the following tables are subject to change.

D.C. CHARACTERISTICS (T_A = 0°C to 70°C; V_{CC} = 5V ±20%; V_{SS} = 0V)

Symbol	Parameter	Min	Typ(3)	Max	Unit	Test Conditions
V_{IL}	Input Low Voltage (Except $\overline{EA}$)	−0.5		0.2 V_{CC} −0.1	V	
V_{IL1}	Input Low Voltage ($\overline{EA}$)	−0.5		0.2 V_{CC} − 0.3	V	
V_{IH}	Input High Voltage (Except XTAL1, RST)	0.2 V_{CC} + 0.9		V_{CC} + 0.5	V	
V_{IH1}	Input High Voltage (XTAL1, RST)	0.7 V_{CC}		V_{CC} + 0.5	V	
V_{OL}	Output Low Voltage (Ports 1, 2, 3)			0.45	V	I_{OL} = 1.6 mA (1)
V_{OL1}	Output Low Voltage (Port 0, ALE, $\overline{PSEN}$)			0.45	V	I_{OL} = 3.2 mA (1)
V_{OH}	Output High Voltage (Ports 1, 2, 3, ALE, $\overline{PSEN}$)	2.4			V	I_{OH} = −60 μA V_{CC} = 5V ±10%
		0.9 V_{CC}			V	I_{OH} = −10 μA
V_{OH1}	Output High Voltage (Port 0 in External Bus Mode)	2.4			V	I_{OH} = −400 μA V_{CC} = 5V ±10%
		0.9 V_{CC}			V	I_{OH} = −40 μA (2)
I_{IL}	Logical 0 Input Current (Ports 1, 2, 3)			−50	μA	V_{IN} = 0.45V
I_{TL}	Logical 1 to 0 Transition Current (Ports 1, 2, 3)			−650	μA	V_{IN} = 2V
I_{LI}	Input Leakage Current (Port 0, $\overline{EA}$)			±10	μA	0.45 < V_{IN} < V_{CC}
RRST	Reset Pulldown Resistor	50		150	KΩ	
CIO	Pin Capacitance			10	pF	Test Freq = 1 MHz, T_A = 25°C
I_{CC}	Power Supply Current:					
	Active Mode, 12 MHz (4)		11	20	mA	
	Idle Mode, 12 MHz (4)		1.7	5	mA	(5)
	Power Down Mode		5	50	μA	

NOTES:
1. Capacitive loading on Ports 0 and 2 may cause spurious noise pulses to be superimposed on the V_{OLS} of ALE and Ports 1 and 3. The noise is due to external bus capacitance discharging into the Port 0 and Port 2 pins when these pins make 1-to-0 transitions during bus operations. In the worst cases (capacitive loading > 100 pF), the noise pulse on the ALE line may exceed 0.8V. In such cases it may be desirable to qualify ALE with a Schmitt Trigger, or use an address latch with a Schmitt Trigger STROBE input.
2. Capacitive loading on Ports 0 and 2 may cause the V_{OH} on ALE and $\overline{PSEN}$ to momentarily fall below the 0.9 V_{CC} specification when the address bits are stabilizing.
3. "Typicals" are based on a limited number of samples taken from early manufacturing lots and are not guaranteed. The values listed are at room temperature, 5V.
4. ICCMAX at other frequencies is given by
   Active Mode: ICCMAX = 1.47 × FREQ + 2.35
   Idle Mode: ICCMAX = 0.33 × FREQ + 1.05
where FREQ is the external oscillator frequency in MHz. ICCMAX is given in mA. See Figure 5.
5. See Figures 6 through 9 for ICC test conditions.

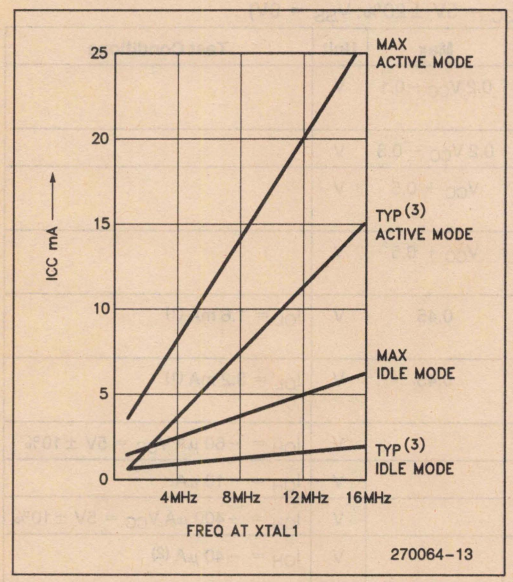


Figure 5. I_{CC} vs. Frequency. Valid only within frequency specifications of the device under test.

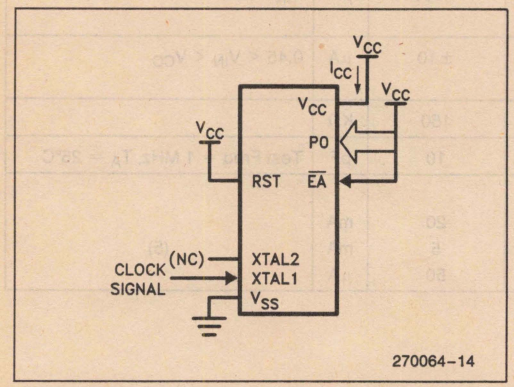


Figure 6. I_{CC} Test Condition, Active Mode. All other pins are disconnected.

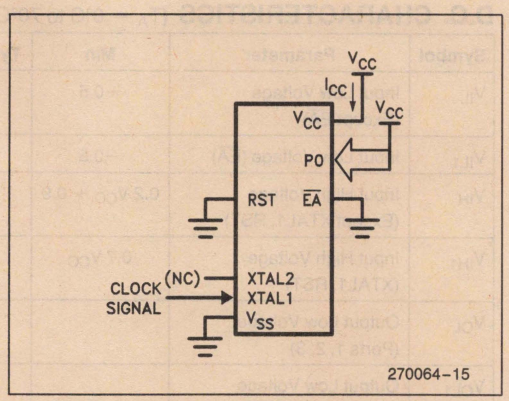


Figure 7. I_{CC} Test Condition, Idle Mode. All other pins are disconnected.

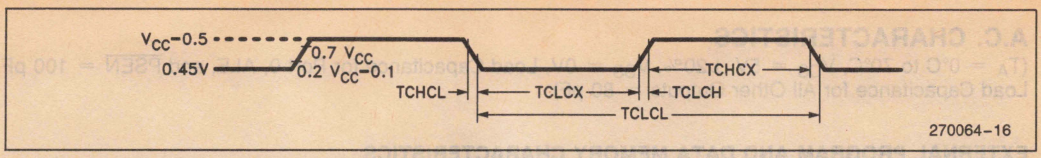


Figure 8. Clock Signal Waveform for I_{CC} Tests in Active and Idle Modes. TCLCH = TCHCL = 5 ns.

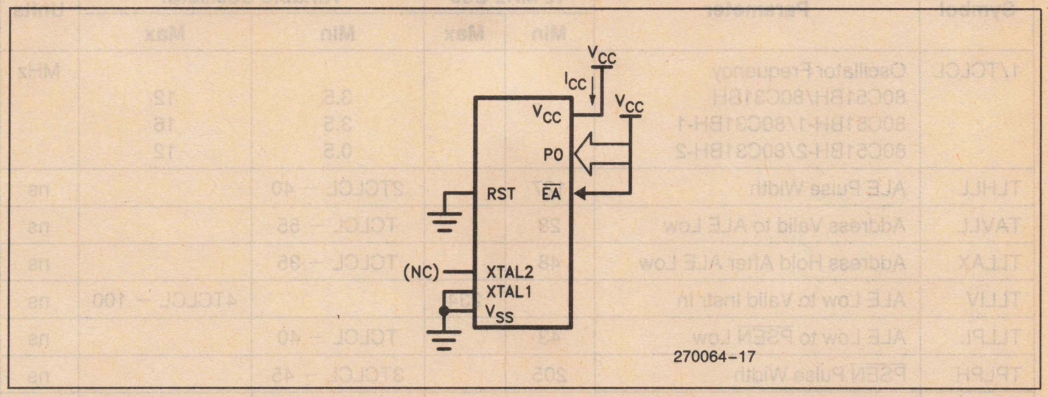


Figure 9. I_{CC} Test Condition, Power Down Mode. All other pins are disconnected.

EXPLANATION OF THE AC SYMBOLS

Each timing symbol has 5 characters. The first character is always a 'T' (stands for time). The other characters, depending on their positions, stand for the name of a signal or the logical status of that signal. The following is a list of all the characters and what they stand for.

A: Address.
C: Clock.
D: Input data.
H: Logic level HIGH.
I: Instruction (program memory contents).
L: Logic level LOW, or ALE.
P: $\overline{PSEN}$.
Q: Output data.
R: $\overline{RD}$ signal.
T: Time.
V: Valid.
W: $\overline{WR}$ signal.
X: No longer a valid logic level.
Z: Float.

EXAMPLE:

TAVLL = Time for Address Valid to ALE Low.
TLLPL = Time for ALE Low to $\overline{PSEN}$ Low.

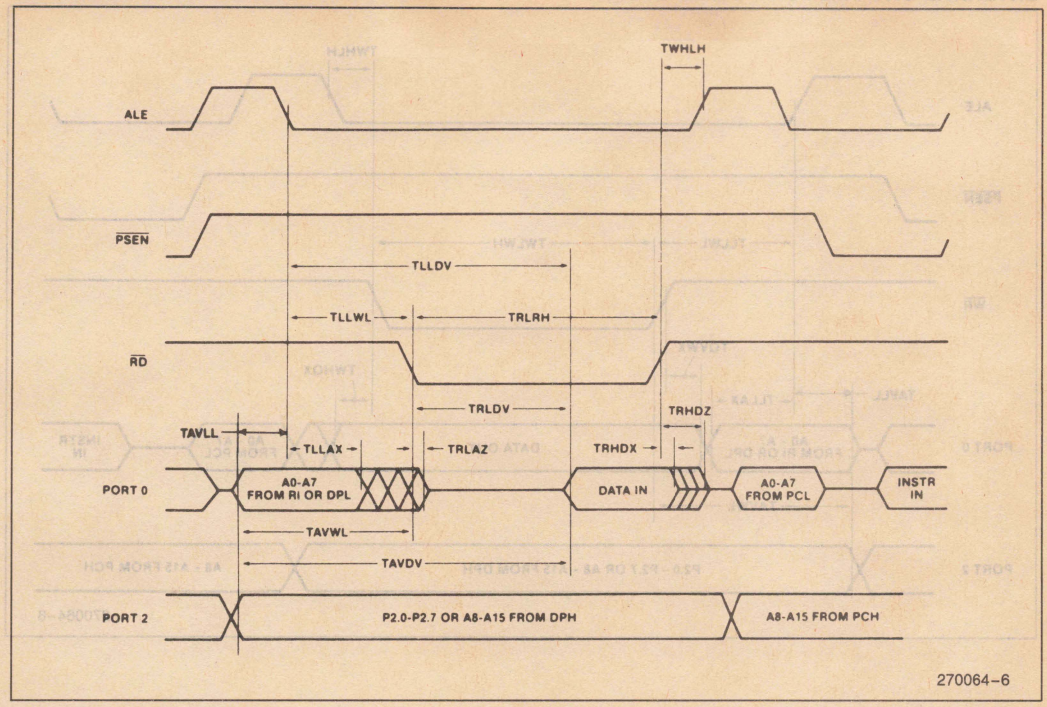
A.C. CHARACTERISTICS

(T_A = 0°C to 70°C, V_{CC} = 5V ±20%, V_{SS} = 0V, Load Capacitance for Port 0, ALE, and $\overline{PSEN}$ = 100 pF, Load Capacitance for All Other Outputs = 80 pF)

EXTERNAL PROGRAM AND DATA MEMORY CHARACTERISTICS

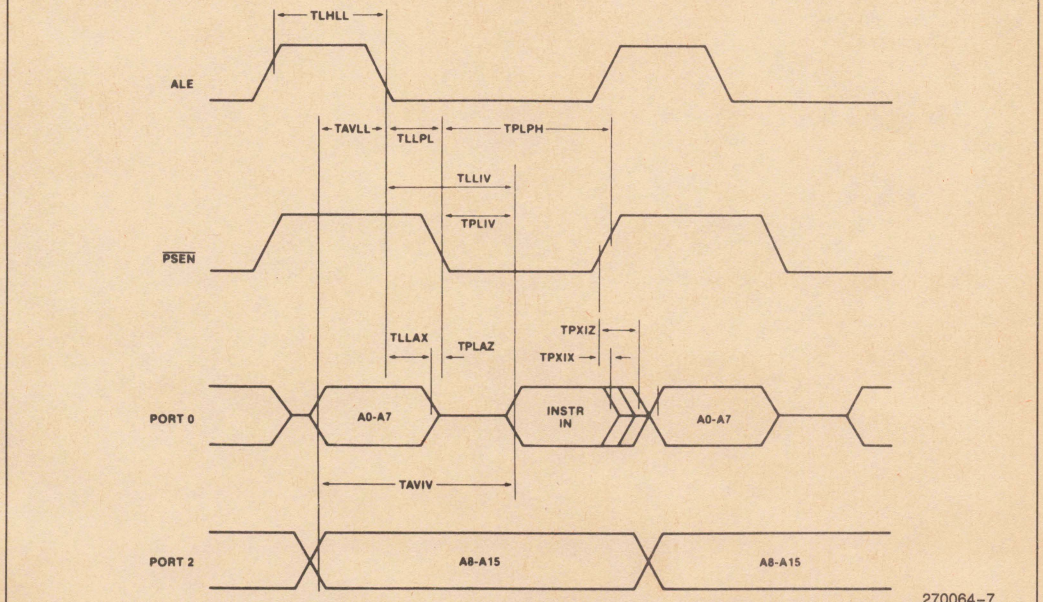
Symbol	Parameter	12 MHz Osc		Variable Oscillator		Units
		Min	Max	Min	Max	
1/TCLCL	Oscillator Frequency 80C51BH/80C31BH 80C51BH-1/80C31BH-1 80C51BH-2/80C31BH-2			 3.5 3.5 0.5	 12 16 12	MHz
TLHLL	ALE Pulse Width	127		2TCLCL − 40		ns
TAVLL	Address Valid to ALE Low	28		TCLCL − 55		ns
TLLAX	Address Hold After ALE Low	48		TCLCL − 35		ns
TLLIV	ALE Low to Valid Instr In		234		4TCLCL − 100	ns
TLLPL	ALE Low to $\overline{PSEN}$ Low	43		TCLCL − 40		ns
TPLPH	$\overline{PSEN}$ Pulse Width	205		3TCLCL − 45		ns
TPLIV	$\overline{PSEN}$ Low to Valid Instr In		145		3TCLCL − 105	ns
TPXIX	Input Instr Hold After $\overline{PSEN}$	0		0		ns
TPXIZ	Input Instr Float After $\overline{PSEN}$		59		TCLCL − 25	ns
TAVIV	Address to Valid Instr In		312		5TCLCL − 105	ns
TPLAZ	$\overline{PSEN}$ Low to Address Float		10		10	ns
TRLRH	$\overline{RD}$ Pulse Width	400		6TCLCL − 100		ns
TWLWH	$\overline{WR}$ Pulse Width	400		6TCLCL − 100		ns
TRLDV	$\overline{RD}$ Low to Valid Data In		252		5TCLCL − 165	ns
TRHDX	Data Hold After $\overline{RD}$	0		0		ns
TRHDZ	Data Float After $\overline{RD}$		97		2TCLCL − 70	ns
TLLDV	ALE Low to Valid Data In		517		8TCLCL − 150	ns
TAVDV	Address to Valid Data In		585		9TCLCL − 165	ns
TLLWL	ALE Low to $\overline{RD}$ or $\overline{WR}$ Low	200	300	3TCLCL − 50	3TCLCL + 50	ns
TAVWL	Address Valid to $\overline{RD}$ or $\overline{WR}$ Low	203		4TCLCL − 130		ns
TQVWX	Data Valid to $\overline{WR}$ Transition	23		TCLCL − 60		ns
TWHQX	Data Hold After $\overline{WR}$	33		TCLCL − 50		ns
TRLAZ	$\overline{RD}$ Low to Address Float		0		0	ns
TWHLH	$\overline{RD}$ or $\overline{WR}$ High to ALE High	43	123	TCLCL − 40	TCLCL + 40	ns

EXTERNAL DATA MEMORY READ CYCLE



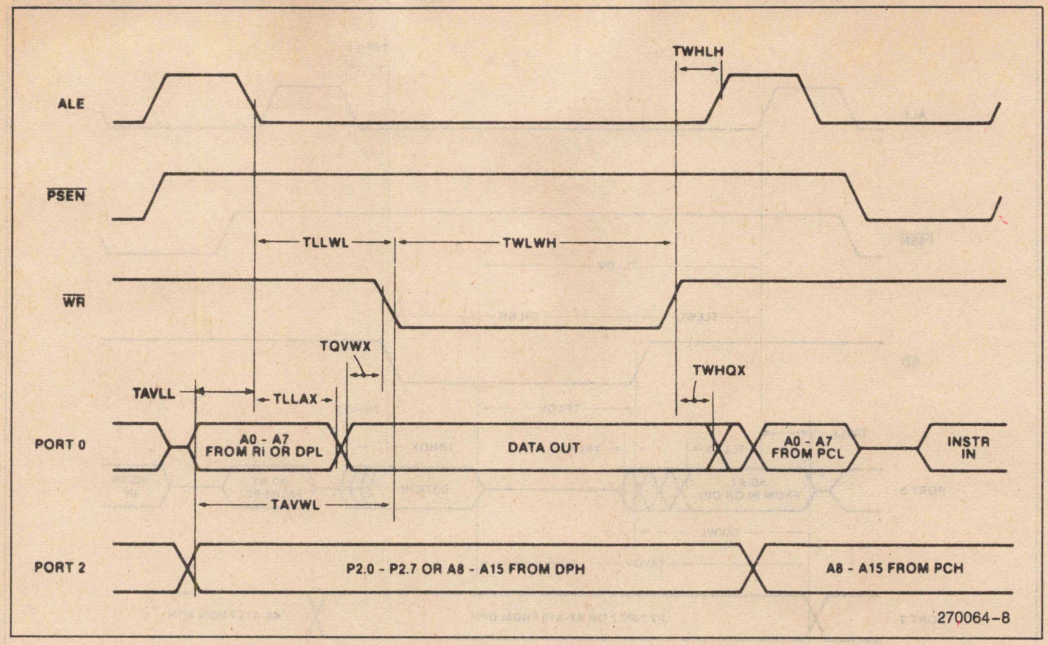
270064-6

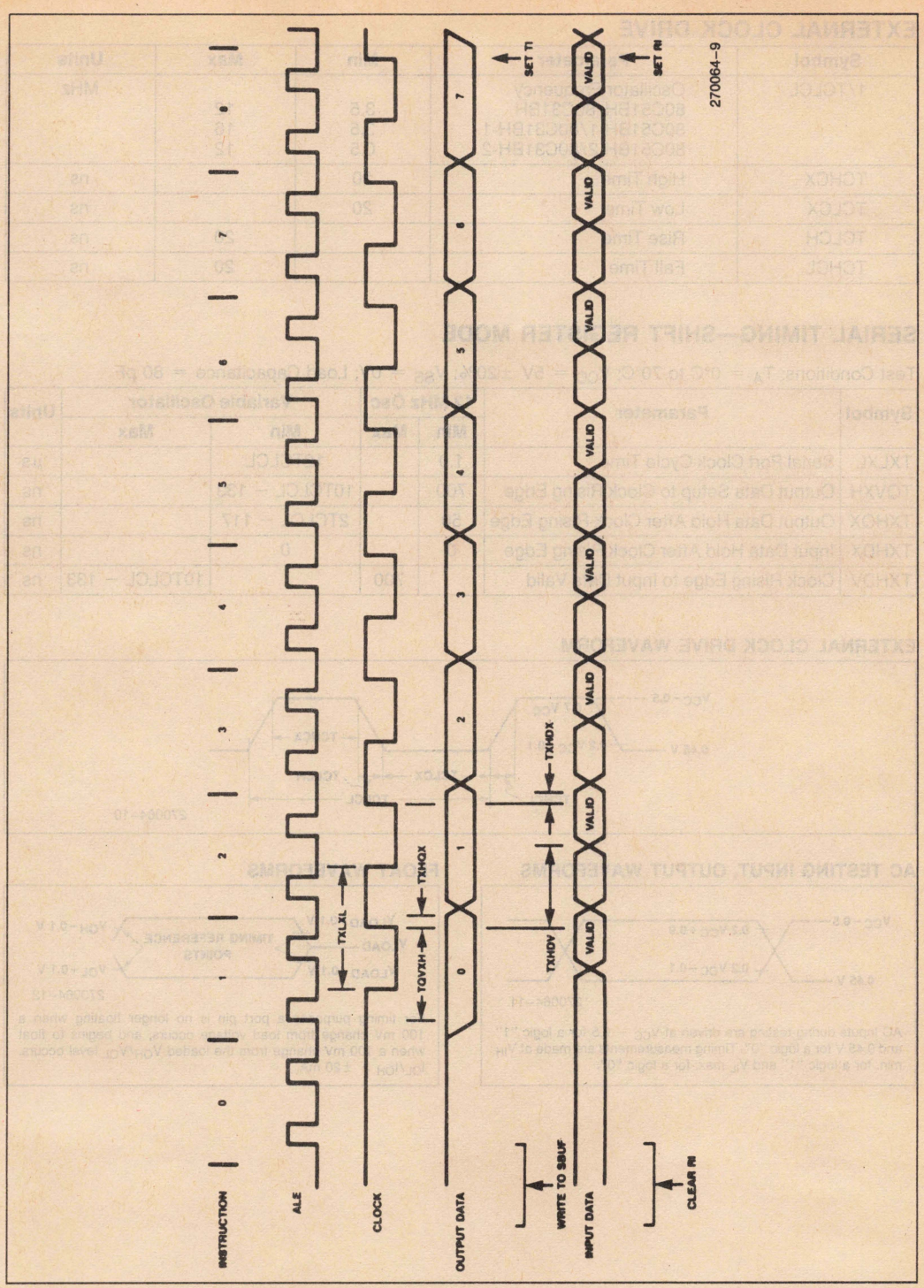
EXTERNAL PROGRAM MEMORY READ CYCLE



270064-7

EXTERNAL DATA MEMORY WRITE CYCLE





Shift Register Mode Timing Waveforms

EXTERNAL CLOCK DRIVE

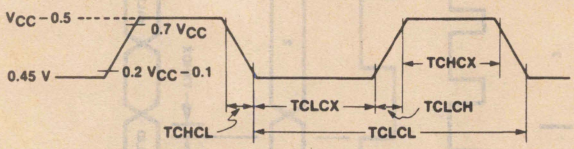
Symbol	Parameter	Min	Max	Units
1/TCLCL	Oscillator Frequency 80C51BH/80C31BH 80C51BH-1/80C31BH-1 80C51BH-2/80C31BH-2	 3.5 3.5 0.5	 12 16 12	MHz
TCHCX	High Time	20		ns
TCLCX	Low Time	20		ns
TCLCH	Rise Time		20	ns
TCHCL	Fall Time		20	ns

SERIAL TIMING—SHIFT REGISTER MODE

Test Conditions: T_A = 0°C to 70°C; V_{CC} = 5V ±20%; V_{SS} = 0V; Load Capacitance = 80 pF

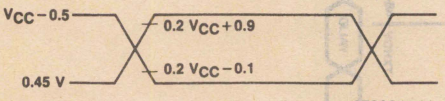
Symbol	Parameter	12 MHz Osc		Variable Oscillator		Units
		Min	Max	Min	Max	
TXLXL	Serial Port Clock Cycle Time	1.0		12TCLCL		μs
TQVXH	Output Data Setup to Clock Rising Edge	700		10TCLCL − 133		ns
TXHQX	Output Data Hold After Clock Rising Edge	50		2TCLCL − 117		ns
TXHDX	Input Data Hold After Clock Rising Edge	0		0		ns
TXHDV	Clock Rising Edge to Input Data Valid		700		10TCLCL − 133	ns

EXTERNAL CLOCK DRIVE WAVEFORM



270064-10

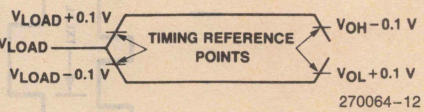
AC TESTING INPUT, OUTPUT WAVEFORMS



270064-11

AC Inputs during testing are driven at V_{CC} − 0.5 for a logic "1" and 0.45 V for a logic "0". Timing measurements are made at V_{IH} min. for a logic "1" and V_{IL} max. for a logic "0".

FLOAT WAVEFORMS



270064-12

For timing purposes a port pin is no longer floating when a 100 mV change from load voltage occurs, and begins to float when a 100 mV change from the loaded V_{OH}/V_{OL} level occurs. $I_{OL}/I_{OH} \geq \pm 20$ mA.

87C51
CHMOS SINGLE-CHIP 8-BIT MICROCONTROLLER WITH 4K BYTES OF EPROM PROGRAM MEMORY

- High Performance CHMOS EPROM
- Quick-Pulse EPROM Programming
- 2-Level Program Memory Lock
- Boolean Processor
- 128-Byte Data RAM
- 32 Programmable I/O Lines
- Two 16-Bit Timer/Counters
- 5 Interrupt Sources
- Programmable Serial Channel
- TTL- and CMOS-Compatible Logic Levels
- 64K External Program Memory Space
- 64K External Data Memory Space
- IDLE and POWER DOWN Modes
- ONCE Mode Facilitates System Testing
- LCC and DIP Packaging Available

The 87C51 is the EPROM version of the 80C51BH. It is fabricated on Intel's CHMOS II-E process. It contains 4K bytes of on-chip Program memory that can be electrically programmed, and can be erased by exposure to ultraviolet light.

The 87C51 EPROM array uses a modified Quick-Pulse programming algorithm, by which the entire 4K-byte array can be programmed in about 12 seconds.

The extremely low operating power, along with the two reduced power modes, Idle and Power Down, make this part very suitable for low power applications. The Idle mode freezes the CPU while allowing the RAM, timer/counters, serial port, and interrupt system to continue functioning. The Power Down mode saves the RAM contents but freezes the oscillator, causing all other chip functions to be inoperative.

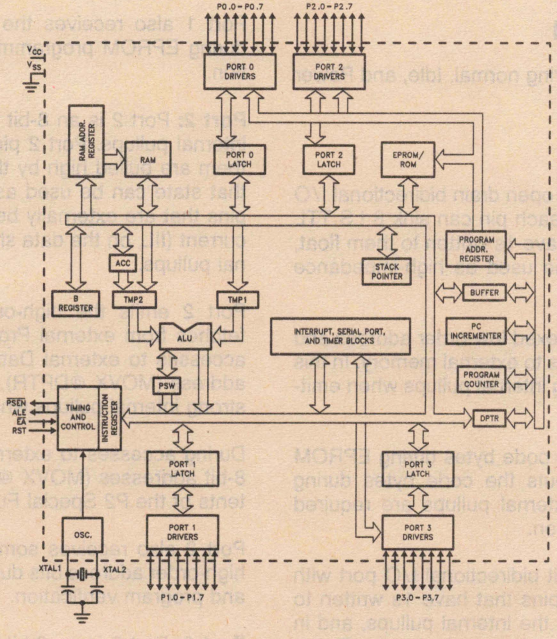


Figure 1. MCS®-51 Architectural Block Diagram

November 1986
Order Number: 270147-002

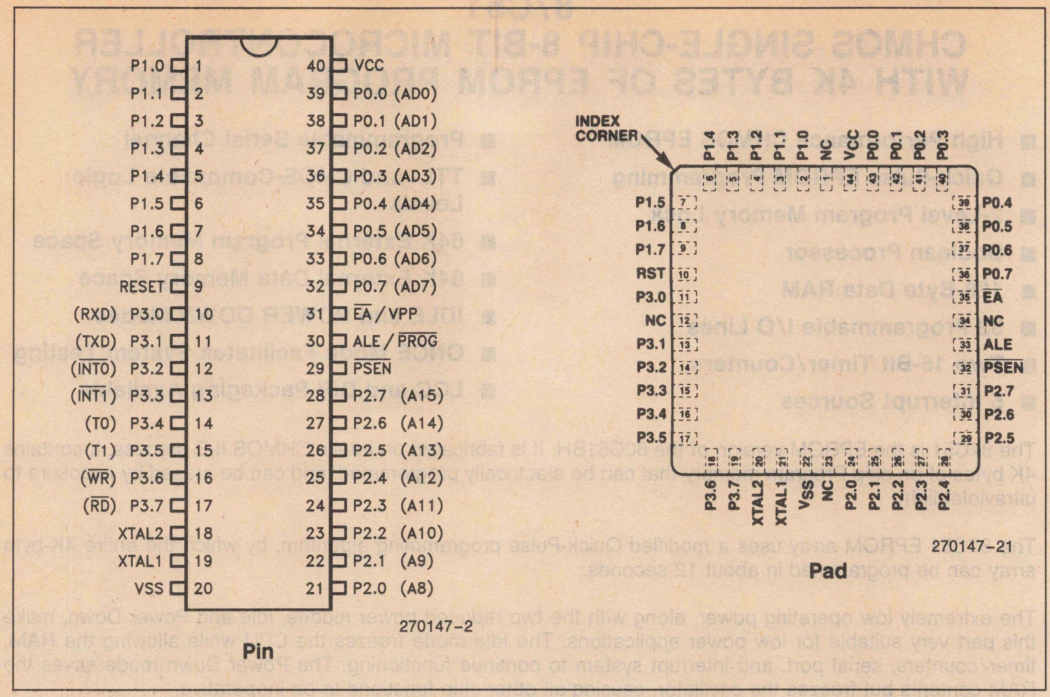


Figure 2. Pin Connections

PIN DESCRIPTION

VCC: Supply voltage during normal, Idle, and Power Down operations.

VSS: Circuit ground.

Port 0: Port 0 is an 8-bit open drain bidirectional I/O port. As an output port each pin can sink 8 LS TTL inputs. Port 0 pins that have 1s written to them float, and in that state can be used as high-impedance inputs.

Port 0 is also the multiplexed low-order address and data bus during accesses to external memory. In this application it uses strong internal pullups when emitting 1s.

Port 0 also receives the code bytes during EPROM programming, and outputs the code bytes during program verification. External pullups are required during program verification.

Port 1: Port 1 is an 8-bit bidirectional I/O port with internal pullups. Port 1 pins that have 1s written to them are pulled high by the internal pullups, and in that state can be used as inputs. As inputs, Port 1 pins that are externally being pulled low will source current (IIL, on the data sheet) because of the internal pullups.

Port 1 also receives the low-order address bytes during EPROM programming and program verification.

Port 2: Port 2 is an 8-bit bidirectional I/O port with internal pullups. Port 2 pins that have 1s written to them are pulled high by the internal pullups, and in that state can be used as inputs. As inputs, Port 2 pins that are externally being pulled low will source current (IIL, on the data sheet) because of the internal pullups.

Port 2 emits the high-order address byte during fetches from external Program memory and during accesses to external Data Memory that use 16-bit address (MOVX @DPTR). In this application it uses strong internal pullups when emitting 1s.

During accesses to external Data Memory that use 8-bit addresses (MOVX @Ri), Port 2 emits the contents of the P2 Special Function Register.

Port 2 also receives some control signals and the high-order address bits during EPROM programming and program verification.

Port 3: Port 3 is an 8-bit bidirectional I/O port with internal pullups. Port 3 pins that have 1s written to them are pulled high by the internal pullups, and in

that state can be used as inputs. As inputs, Port 3 pins that are externally being pulled low will source current (IIL, on the data sheet) because of the pull-ups.

Port 3 also serves the functions of various special features of the MCS-51 Family, as listed below:

Pin	Name	Alternate Function
P3.0	RXD	Serial input line
P3.1	TXD	Serial output line
P3.2	$\overline{INT0}$	External Interrupt 0
P3.3	$\overline{INT1}$	External Interrupt 1
P3.4	T0	Timer 0 external input
P3.5	T1	Timer 1 external input
P3.6	$\overline{WR}$	External Data Memory Write strobe
P3.7	$\overline{RD}$	External Data Memory Read strobe

Port 3 also receives some control signals for EPROM programming and program verification.

RST: Reset input. A logic high on this pin for two machine cycles while the oscillator is running resets the device. An internal pulldown resistor permits a power-on reset to be generated using only an external capacitor to VCC.

ALE/$\overline{PROG}$: Address Latch Enable output signal for latching the low byte of the address during accesses to external memory. This pin is also the program pulse input ($\overline{PROG}$) during EPROM programming.

In normal operation ALE is emitted at a constant rate of 1/6 the oscillator frequency, and may be used for external timing or clocking purposes. Note, however, that one ALE pulse is skipped during each access to external Data Memory.

$\overline{PSEN}$: Program Store Enable is the Read strobe to External Program Memory. When the 87C51 is executing from Internal Program Memory, $\overline{PSEN}$ is inactive (high). When the device is executing code from External Program Memory, $\overline{PSEN}$ is activated twice each machine cycle, except that two $\overline{PSEN}$ activations are skipped during each access to External Data Memory.

$\overline{EA}$/VPP: External Access enable. $\overline{EA}$ must be strapped to V_{SS} in order to enable the 87C51 to fetch code from External Program Memory locations 0000H to 0FFFH. Note, however, that if either of the Lock Bits is programmed, the logic level at $\overline{EA}$ is internally latched during reset.

$\overline{EA}$ must be strapped to V_{CC} for internal program execution.

This pin also receives the 12.75V programming supply voltage (VPP) during EPROM programming.

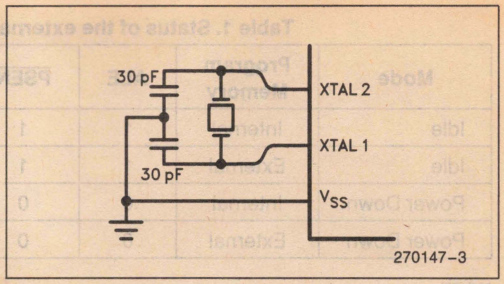


Figure 3. Using the On-Chip Oscillator

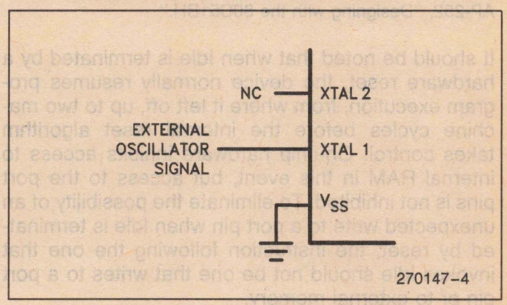


Figure 4. External Clock Drive

XTAL1: Input to the inverting oscillator amplifier and input to the internal clock generating circuits.

XTAL2: Output from the inverting oscillator amplifier.

OSCILLATOR CHARACTERISTICS

XTAL1 and XTAL2 are the input and output, respectively, of an inverting amplifier which can be configured for use as an on-chip oscillator, as shown in Figure 3.

To drive the device from an external clock source, XTAL1 should be driven, while XTAL2 is left unconnected, as shown in Figure 4. There are no requirements on the duty cycle of the external clock signal, since the input to the internal clocking circuitry is through a divide-by-two flip-flop, but minimum and maximum high and low times specified on the Data Sheet must be observed.

IDLE MODE

In Idle Mode, the CPU puts itself to sleep while all the on-chip peripherals remain active. The mode is invoked by software. The content of the on-chip RAM and all the Special Functions Registers remain unchanged during this mode. The Idle Mode can be terminated by any enabled interrupt or by a hardware reset.

Table 1. Status of the external pins during Idle and Power Down

Mode	Program Memory	ALE	$\overline{PSEN}$	PORT0	PORT1	PORT2	PORT3
Idle	Internal	1	1	Data	Data	Data	Data
Idle	External	1	1	Float	Data	Address	Data
Power Down	Internal	0	0	Data	Data	Data	Data
Power Down	External	0	0	Float	Data	Data	Data

NOTE:
For more detailed information on the reduced power modes refer to current Microcontroller Handbook, and Application Note AP-252, "Designing with the 80C51BH."

It should be noted that when Idle is terminated by a hardware reset, the device normally resumes program execution, from where it left off, up to two machine cycles before the internal reset algorithm takes control. On-chip hardware inhibits access to internal RAM in this event, but access to the port pins is not inhibited. To eliminate the possibility of an unexpected write to a port pin when Idle is terminated by reset, the instruction following the one that invokes Idle should not be one that writes to a port pin or to external memory.

POWER DOWN MODE

In Power Down Mode, the oscillator is stopped and all on-chip functions cease except that the on-chip RAM content is maintained. The mode is invoked by software. The Power Down Mode can be terminated only by a hardware reset.

DESIGN CONSIDERATIONS

Exposure to light when the device is in operation may cause logic errors. For this reason, it is suggested that an opaque label be placed over the window when the die is exposed to ambient light.

PROGRAM MEMORY LOCK

The 87C51 contains two program memory lock schemes: Encrypted Verify and Lock Bits.

Encrypted Verify: The 87C51 implements a 32-byte EPROM array that can be programmed by the customer, and which can then be used to encrypt the program code bytes during EPROM verification. The EPROM verification procedure is performed as usual, except that each code byte comes out logically X-NORed with one of the 32 key bytes. The key bytes are gone through in sequence. Therefore, to read the ROM code, one has to know the 32 key bytes in their proper sequence.

Lock Bits: Also on the chip are two Lock Bits which can be left unprogrammed (U) or can be programmed (P) to obtain the following additional features:

Bit 2	Bit 1	Additional Features
U	U	none
U	P	• Externally fetched code can not access internal Program Memory. • Further programming disabled.
P	U	(Reserved for Future definition.)
P	P	• Externally fetched code can not access internal Program Memory. • Further programming disabled. • Program verification is disabled.

When Lock Bit 1 is programmed, the logic level at the $\overline{EA}$ pin is sampled and latched during reset. If the device is powered up without a reset, the latch initializes to a random value, and holds that value until reset is activated. It is necessary that the latched value of $\overline{EA}$ be in agreement with the current logic level at that pin in order for the device to function properly.

ONCE MODE

The ONCE ("on-circuit emulation") mode facilitates testing and debugging of systems using the 87C51 without the 87C51 having to be removed from the circuit. The ONCE mode is invoked by:

1. Pull ALE low while the device is in reset and $\overline{PSEN}$ is high;
2. Hold ALE low as RST is deactivated.

While the device is in ONCE mode, the Port 0 pins go into a float state, and the other port pins and ALE and $\overline{PSEN}$ are weakly pulled high. The oscillator circuit remains active. While the 87C51 is in this mode, an emulator or test CPU can be used to drive the circuit. Normal operation is restored when a normal reset is applied.

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias .....0°C to 70°C
Storage Temperature .....−65°C to +150°C
Voltage on $\overline{EA}$/VPP Pin to VSS .....0V to +13.0V
Voltage on Any Other Pin to VSS ..−0.5V to +6.5V
Power Dissipation.....1.5W
(Based on package heat transfer limitations, not device power consumption).

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

NOTICE: Specifications contained within the following tables are subject to change.

D.C. CHARACTERISTICS: (TA = 0°C to 70°C; VCC = 5V ±10%; VSS = 0V)

Symbol	Parameter	Min	Typ(1)	Max	Unit	Test Conditions
VIL	Input Low Voltage (Except $\overline{EA}$)	−0.5		.2VCC−.1	V	
VIL1	Input Low Voltage to $\overline{EA}$	0		.2VCC−.3	V	
VIH	Input High Voltage (Except XTAL1, RST)	.2VCC+.9		VCC+.5	V	
VIH1	Input High Voltage (XTAL1, RST)	0.7VCC		VCC+.5	V	
VOL	Output Low Voltage (Ports 1, 2, 3)			0.45	V	IOL = 1.6 mA (2)
VOL1	Output Low Voltage (Port 0, ALE, $\overline{PSEN}$)			0.45	V	IOL = 3.2 mA (2)
VOH	Output High Voltage (Ports 1, 2, 3, ALE, $\overline{PSEN}$)	2.4			V	IOH = −60 µA
		.9VCC			V	IOH = −10 µA
VOH1	Output High Voltage (Port 0 in External Bus Mode)	2.4			V	IOH = −800 µA
		.9VCC			V	IOH = −80 µA (3)
IIL	Logical 0 Input Current (Ports 1, 2, 3)			−50	µA	Vin = 0.45 V
ITL	Logical 1-to-0 transition current (Ports 1, 2, 3)			−650	µA	(4)
ILI	Input Leakage Current (Port 0)			±10	µA	Vin = VIL or VIH
ICC	Power Supply Current: Active Mode @ 12 MHz (5) Idle Mode @ 12 MHz (5) Power Down Mode		 11.5 1.3 3	 25 4 50	 mA mA µA	 (6)
RRST	Internal Reset Pulldown Resistor	50		300	kΩ	
CIO	Pin Capacitance			10	pF	

NOTES:
1. "Typicals" are based on a limited number of samples taken from early manufacturing lots and are not guaranteed. The values listed are at room temp, 5V.
2. Capacitive loading on Ports 0 and 2 may cause spurious noise pulses to be superimposed on the VOLs of ALE and Ports 1 and 3. The noise is due to external bus capacitance discharging into the Port 0 and Port 2 pins when these pins make 1-to-0 transitions during bus operations. In the worst cases (capacitive loading > 100pF), the noise pulse on the ALE pin may exceed 0.8V. In such cases it may be desirable to qualify ALE with a Schmitt Trigger, or use an address latch with a Schmitt Trigger STROBE input.
3. Capacitive loading on Ports 0 and 2 may cause the VOH on ALE and $\overline{PSEN}$ to momentarily fall below the .9VCC specification when the address bits are stabilizing.
4. Pins of Ports 1, 2, and 3 source a transition current when they are being externally driven from 1 to 0. The transition current reaches its maximum value when Vin is approximately 2V.
5. ICCMAX at other frequencies is given by:

Active Mode: ICCMAX = 0.94 × FREQ + 13.71
Idle Mode: ICCMAX = 0.14 × FREQ + 2.31

where FREQ is the external oscillator frequency in MHz. ICCMAX is given in mA. See Figure 5.
6. See Figures 6 through 9 for ICC test conditions.

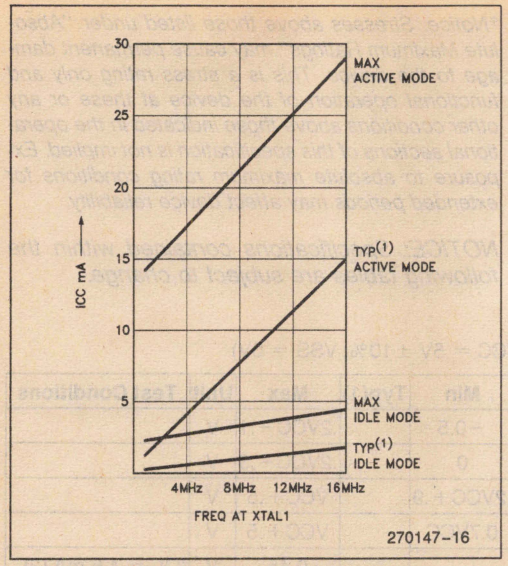


Figure 5. ICC vs. FREQ. Valid only within frequency specifications of the device under test.

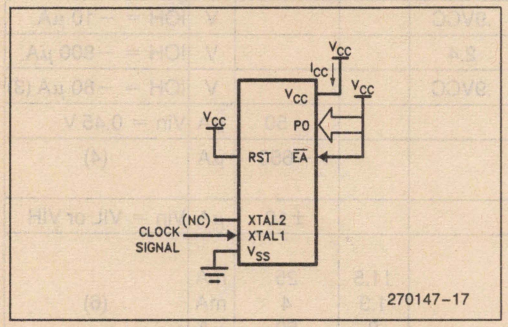


Figure 6. ICC Test Condition, Active Mode. All other pins are disconnected.

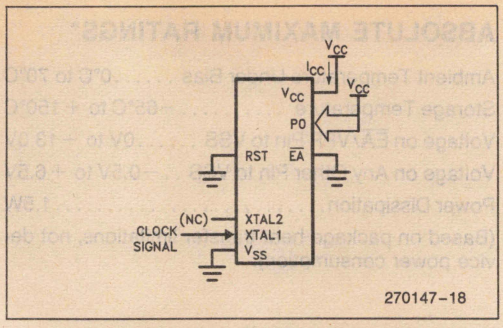


Figure 7. ICC Test Condition, Idle Mode. All other pins are disconnected.

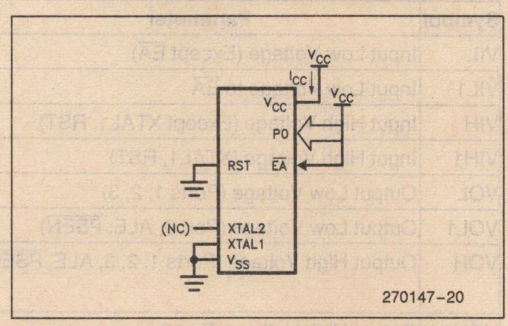


Figure 9. ICC Test Condition, Power Down Mode. All other pins are disconnected.

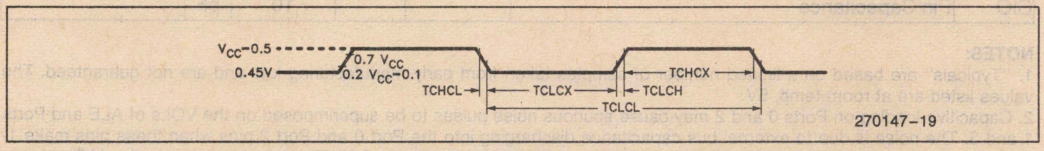


Figure 8. Clock Signal Waveform for ICC tests in Active and Idle Modes. TCLCH = TCHCL = 5 ns.

EXPLANATION OF THE AC SYMBOLS

Each timing symbol has 5 characters. The first character is always a 'T' (stands for time). The other characters, depending on their positions, stand for the name of a signal or the logical status of that signal. The following is a list of all the characters and what they stand for.

A:Address.
C:Clock.
D:Input data.
H:Logic level HIGH.
I:Instruction (program memory contents).
L:Logic level LOW, or ALE.
P:$\overline{\text{PSEN}}$.
Q:Output data.
R:$\overline{\text{RD}}$ signal.
T:Time.
V:Valid.
W:$\overline{\text{WR}}$ signal.
X:No longer a valid logic level.
Z:Float.

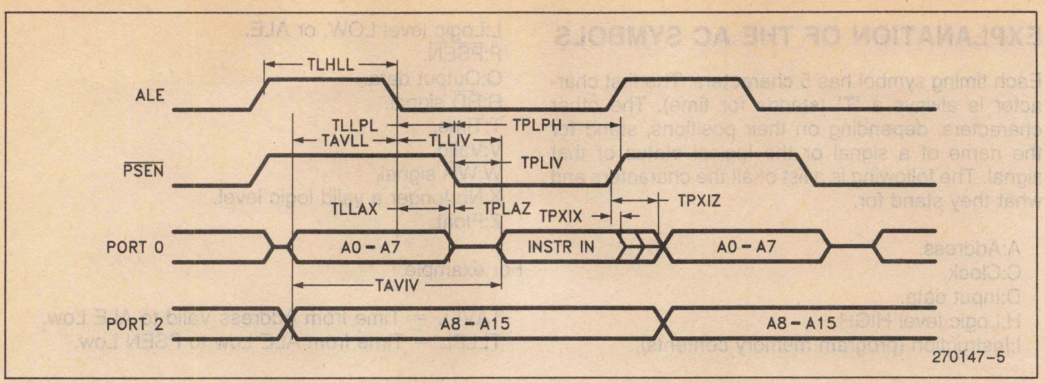
For example,

TAVLL = Time from Address Valid to ALE Low.
TLLPL = Time from ALE Low to PSEN Low.

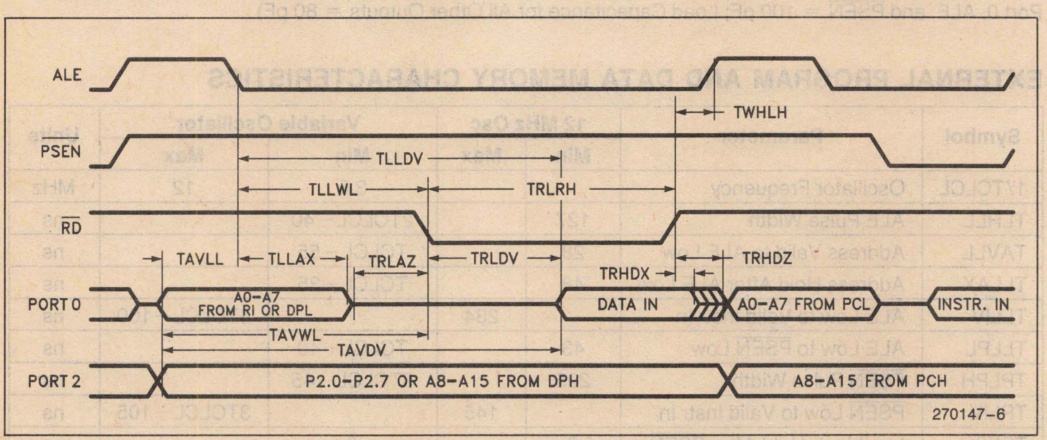
A.C. CHARACTERISTICS: (TA = 0°C to 70°C; VCC = 5V ±10%; VSS = 0V; Load Capacitance for Port 0, ALE, and $\overline{\text{PSEN}}$ = 100 pF; Load Capacitance for All Other Outputs = 80 pF)

EXTERNAL PROGRAM AND DATA MEMORY CHARACTERISTICS

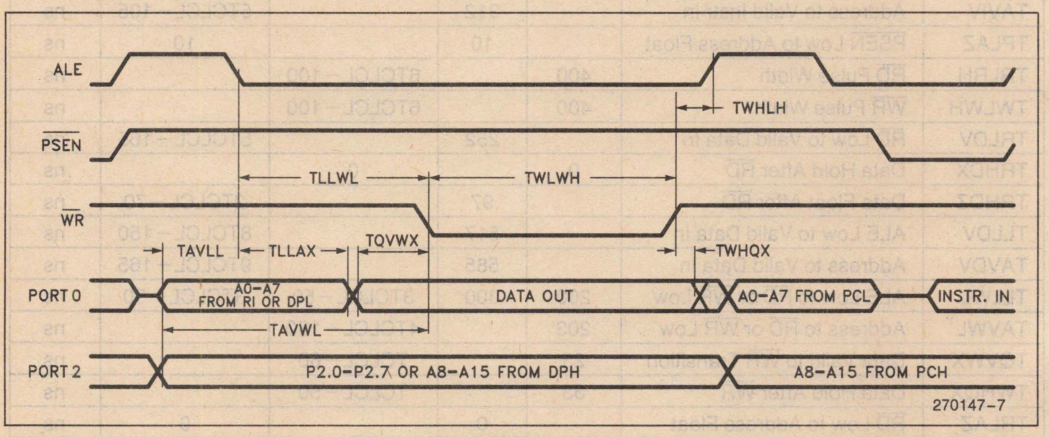
Symbol	Parameter	12 MHz Osc		Variable Oscillator		Units
		Min	Max	Min	Max	
1/TCLCL	Oscillator Frequency			3.5	12	MHz
TLHLL	ALE Pulse Width	127		2TCLCL−40		ns
TAVLL	Address Valid to ALE Low	28		TCLCL−55		ns
TLLAX	Address Hold After ALE Low	48		TCLCL−35		ns
TLLIV	ALE Low to Valid Instr In		234		4TCLCL−100	ns
TLLPL	ALE Low to $\overline{\text{PSEN}}$ Low	43		TCLCL−40		ns
TPLPH	$\overline{\text{PSEN}}$ Pulse Width	205		3TCLCL−45		ns
TPLIV	$\overline{\text{PSEN}}$ Low to Valid Instr In		145		3TCLCL−105	ns
TPXIX	Input Instr Hold After $\overline{\text{PSEN}}$	0		0		ns
TPXIZ	Input Instr Float After $\overline{\text{PSEN}}$		59		TCLCL−25	ns
TAVIV	Address to Valid Instr In		312		5TCLCL−105	ns
TPLAZ	$\overline{\text{PSEN}}$ Low to Address Float		10		10	ns
TRLRH	$\overline{\text{RD}}$ Pulse Width	400		6TCLCL−100		ns
TWLWH	$\overline{\text{WR}}$ Pulse Width	400		6TCLCL−100		ns
TRLDV	$\overline{\text{RD}}$ Low to Valid Data In		252		5TCLCL−165	ns
TRHDX	Data Hold After $\overline{\text{RD}}$	0		0		ns
TRHDZ	Data Float After $\overline{\text{RD}}$		97		2TCLCL−70	ns
TLLDV	ALE Low to Valid Data In		517		8TCLCL−150	ns
TAVDV	Address to Valid Data In		585		9TCLCL−165	ns
TLLWL	ALE Low to $\overline{\text{RD}}$ or $\overline{\text{WR}}$ Low	200	300	3TCLCL−50	3TCLCL+50	ns
TAVWL	Address to $\overline{\text{RD}}$ or $\overline{\text{WR}}$ Low	203		4TCLCL−130		ns
TQVWX	Data Valid to $\overline{\text{WR}}$ Transition	23		TCLCL−60		ns
TWHQX	Data Hold After $\overline{\text{WR}}$	33		TCLCL−50		ns
TRLAZ	$\overline{\text{RD}}$ Low to Address Float		0		0	ns
TWHLH	$\overline{\text{RD}}$ or $\overline{\text{WR}}$ High to ALE High	43	123	TCLCL−40	TCLCL+40	ns



External Program Memory Read Cycle



External Data Memory Read Cycle

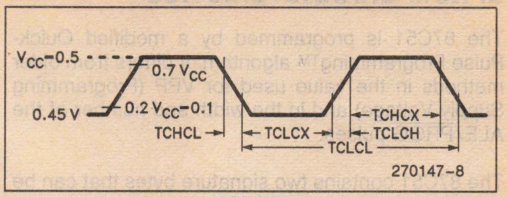


External Data Memory Write Cycle

EXTERNAL CLOCK DRIVE

Symbol	Parameter	Min	Max	Units
1/TCLCL	Oscillator Frequency	3.5	12	MHz
TCHCX	High Time	20		ns
TCLCX	Low Time	20		ns
TCLCH	Rise Time		20	ns
TCHCL	Fall Time		20	ns

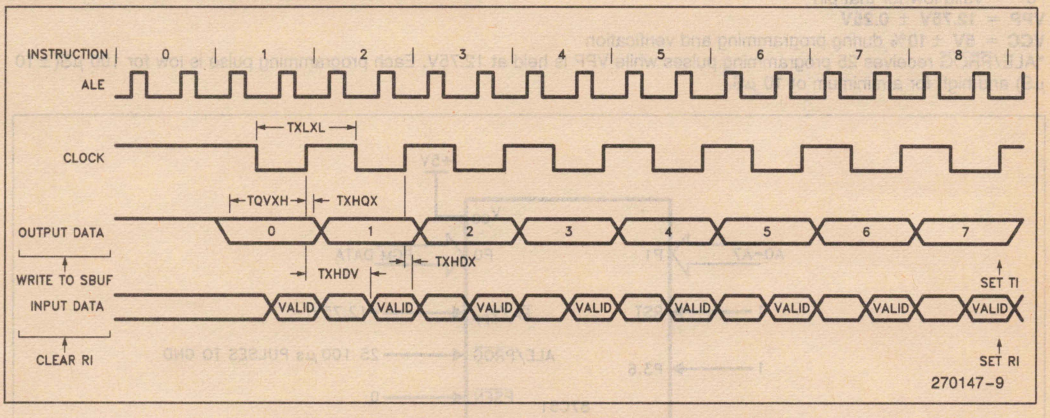
EXTERNAL CLOCK DRIVE WAVEFORM



SERIAL PORT TIMING—SHIFT REGISTER MODE

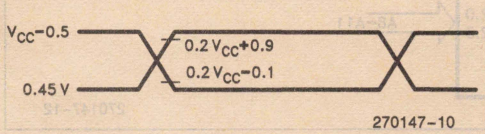
Symbol	Parameter	12 MHz Osc		Variable Oscillator		Units
		Min	Max	Min	Max	
TXLXL	Serial Port Clock Cycle Time	1.0		12TCLCL		μs
TQVXH	Output Data Setup to Clock Rising Edge	700		10TCLCL−133		ns
TXHQX	Output Data Hold After Clock Rising Edge	50		2TCLCL−117		ns
TXHDX	Input Data Hold After Clock Rising Edge	0		0		ns
TXHDV	Clock Rising Edge to Input Data Valid		700		10TCLCL−133	ns

SHIFT REGISTER MODE TIMING WAVEFORMS



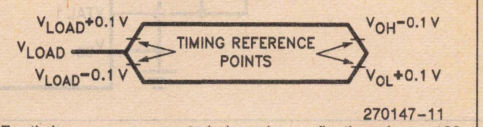
A.C. TESTING:

INPUT, OUTPUT WAVEFORMS



AC inputs during testing are driven at V_{CC} − 0.5 for a Logic "1" and 0.45V for a Logic "0." Timing measurements are made at V_{IH} min for a Logic "1" and V_{IL} max for a Logic "0".

FLOAT WAVEFORM



For timing purposes a port pin is no longer floating when a 100 mV change from load voltage occurs, and begins to float when a 100 mV change from the loaded V_{OH}/V_{OL} level occurs. I_{OL}/I_{OH} ≥ ±20 mA.

EPROM CHARACTERISTICS

The 87C51 is programmed by a modified Quick-Pulse Programming™ algorithm. It differs from older methods in the value used for VPP (Programming Supply Voltage) and in the width and number of the ALE/$\overline{PROG}$ pulses.

The 87C51 contains two signature bytes that can be read and used by an EPROM programming system to identify the device. The signature bytes identify the device as an 87C51 manufactured by Intel.

Table 2 shows the logic levels for reading the signature byte, and for programming the Program Memory, the Encryption Table, and the Lock Bits. The circuit configuration and waveforms for Quick-Pulse Programming™ are shown in Figures 10 and 11. Figure 12 shows the circuit configuration for normal Program Memory verification.

Table 2. EPROM Programming Modes

MODE	RST	$\overline{PSEN}$	ALE/$\overline{PROG}$	$\overline{EA}$/VPP	P2.7	P2.6	P3.7	P3.6
Read Signature	1	0	1	1	0	0	0	0
Program Code Data	1	0	0*	VPP	1	0	1	1
Verify Code Data	1	0	1	1	0	0	1	1
Pgm Encryption Table	1	0	0*	VPP	1	0	1	0
Pgm Lock Bit 1	1	0	0*	VPP	1	1	1	1
Pgm Lock Bit 2	1	0	0*	VPP	1	1	0	0

NOTES:
"1" = Valid high for that pin
"0" = Valid low for that pin
VPP = 12.75V ± 0.25V
VCC = 5V ±10% during programming and verification
*ALE/$\overline{PROG}$ receives 25 programming pulses while VPP is held at 12.75V. Each programming pulse is low for 100 μS(±10 μS) and high for a minimum of 10 μS.

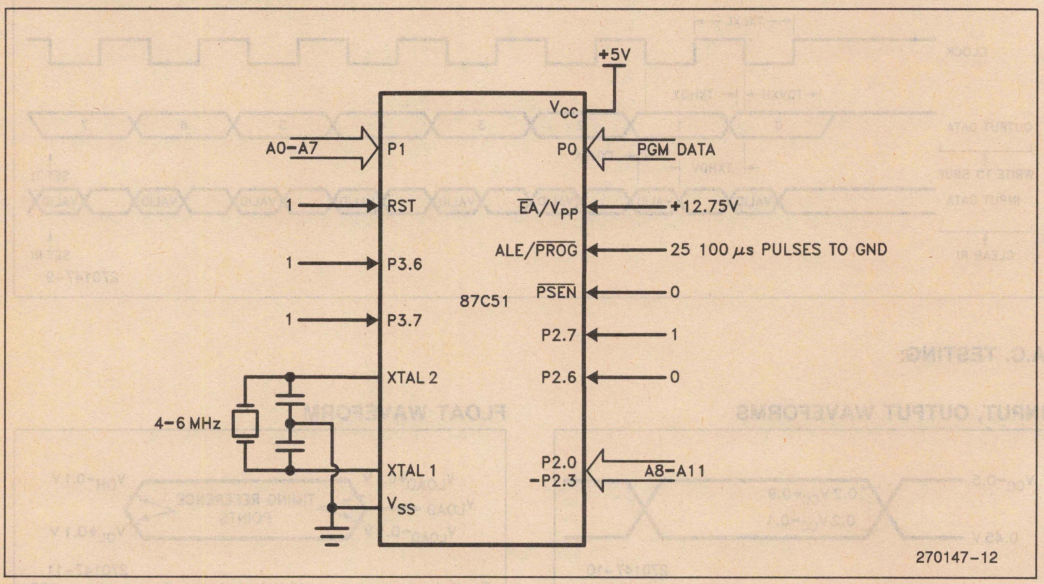


Figure 10. Programming Configuration

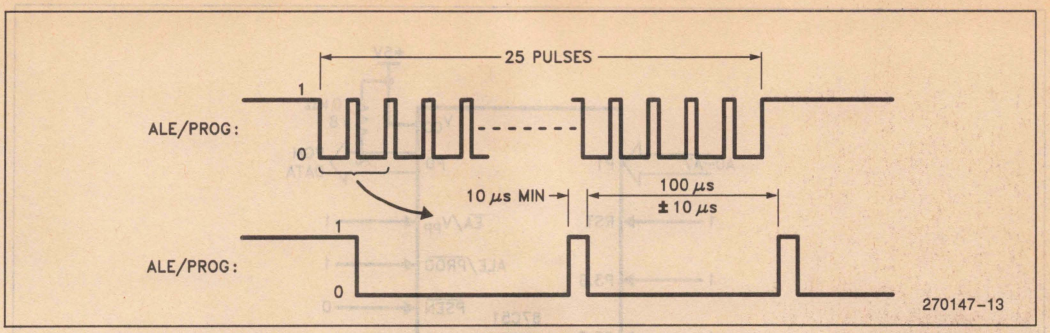


Figure 11. $\overline{\text{PROG}}$ Waveforms

Quick-Pulse Programming™

The setup for Microcontroller Quick-Pulse Programming™ is shown in Figure 10. Note that the 87C51 is running with a 4 to 6 MHz oscillator. The reason the oscillator needs to be running is that the device is executing internal address and program data transfers.

The address of the EPROM location to be programmed is applied to Ports 1 and 2, as shown in Figure 10. The code byte to be programmed into that location is applied to Port 0. RST, $\overline{\text{PSEN}}$, and pins of Ports 2 and 3 specified in Table 2 are held at the "Program Code Data" levels indicated in Table 2. Then ALE/$\overline{\text{PROG}}$ is pulsed low 25 times as shown in Figure 11.

To program the Encryption Table, repeat the 25-pulse programming sequence for addresses 0 through 1FH, using the "Pgm Encryption Table" levels. Don't forget that after the Encryption Table is programmed, verify cycles will produce only encrypted data.

To program the Lock Bits, repeat the 25-pulse programming sequence using the "Pgm Lock Bit" levels. After one Lock Bit is programmed, further programming of the Code Memory and Encryption Table is disabled. However, the other Lock Bit can still be programmed.

Note that the $\overline{\text{EA}}$/VPP pin must not be allowed to go above the maximum specified VPP level for any amount of time. Even a narrow glitch above that voltage level can cause permanent damage to the device. The VPP source should be well regulated and free of glitches and overshoot.

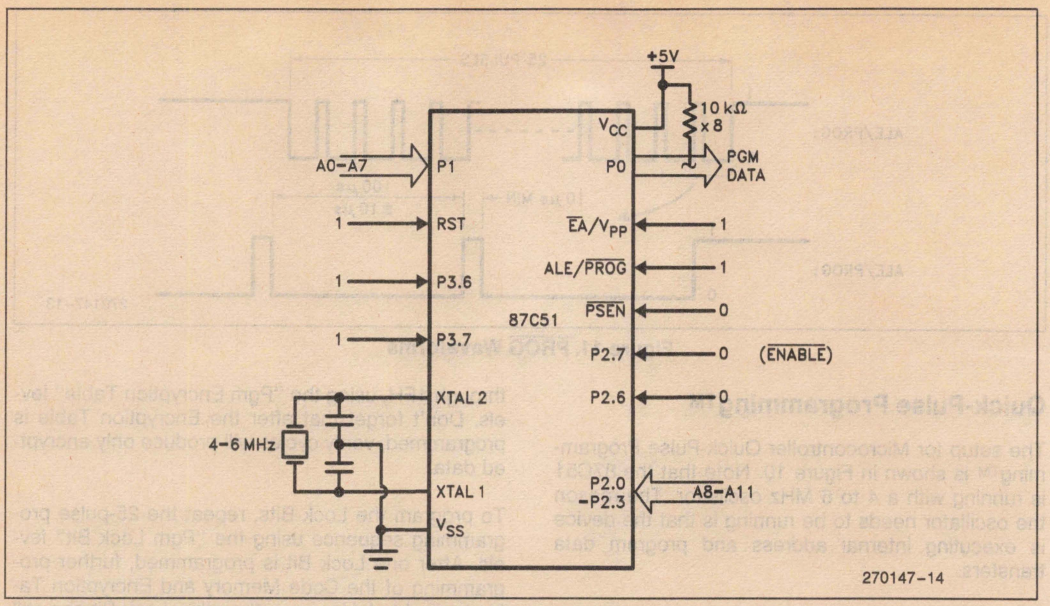


Figure 12. Program Verification

Program Verification

If Lock Bit 2 has not been programmed, the on-chip Program Memory can be read out for program verification. The address of the Program Memory location to be read is applied to Ports 1 and 2 as shown in Figure 12. The other pins are held at the "Verify Code Data" levels indicated in Table 2. The contents of the addressed location will be emitted on Port 0. External pullups are required on Port 0 for this operation. Detailed timing specifications are shown in later sections of this data sheet.

If the Encryption Table has been programmed, the data presented at Port 0 will be the Exclusive NOR of the program byte with one of the encryption bytes. The user will have to know the Encryption Table contents in order to correctly decode the verification data. The Encryption Table itself can not be read out.

Reading the Signature Bytes

The signature bytes are read by the same procedure as a normal verification of locations 030H and 031H, except that P3.6 and P3.7 need to be pulled to a logic low. The values returned are:

(030H) = 89H indicates manufactured by Intel

(031H) = 57H indicates 87C51

Program/Verify Algorithms

Any algorithm in agreement with the conditions listed in Table 2, and which satisfies the timing specifications, is suitable.

Erasure Characteristics

Erasure of the EPROM begins to occur when the chip is exposed to light with wavelengths shorter than approximately 4,000 Angstroms. Since sunlight and fluorescent lighting have wavelengths in this range, exposure to these light sources over an extended time (about 1 week in sunlight, or 3 years in room level fluorescent lighting) could cause inadvertent erasure. If an application subjects the device to this type of exposure, it is suggested that an opaque label be placed over the window.

The recommended erasure procedure is exposure to ultraviolet light (at 2537 Angstroms) to an integrated dose of at least 15 W-sec/cm^2. Exposing the EPROM to an ultraviolet lamp of 12,000 μW/cm^2 rating for 20 to 30 minutes, at a distance of about 1 inch, should be sufficient.

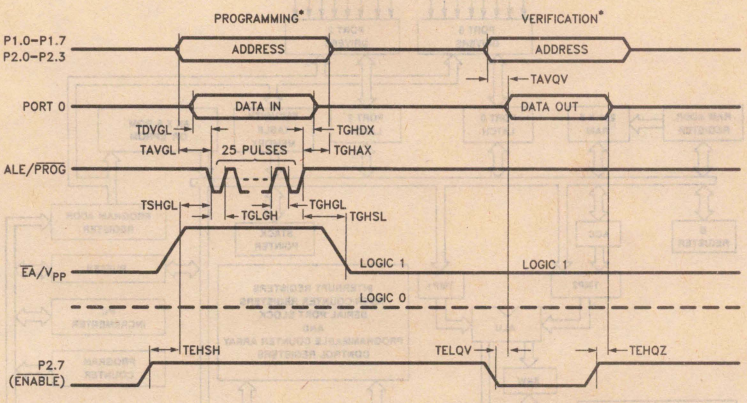
Erasure leaves the array in an all 1s state.

EPROM PROGRAMMING AND VERIFICATION CHARACTERISTICS:

(TA = 21°C to 27°C, VCC = 5V ±10%, VSS = 0V)

Symbol	Parameter	Min	Max	Units
VPP	Programming Supply Voltage	12.5	13.0	V
IPP	Programming Supply Current		50	mA
1/TCLCL	Oscillator Frequency	4	6	MHz
TAVGL	Address Setup to $\overline{PROG}$ Low	48TCLCL		
TGHAX	Address Hold After $\overline{PROG}$	48TCLCL		
TDVGL	Data Setup to $\overline{PROG}$ Low	48TCLCL		
TGHDX	Data Hold After $\overline{PROG}$	48TCLCL		
TEHSH	P2.7 ($\overline{ENABLE}$) High to VPP	48TCLCL		
TSHGL	VPP Setup to $\overline{PROG}$ Low	10		µs
TGHSL	VPP Hold After $\overline{PROG}$	10		µs
TGLGH	$\overline{PROG}$ Width	90	110	µs
TAVQV	Address to Data Valid		48TCLCL	
TELQV	$\overline{ENABLE}$ Low to Data Valid		48TCLCL	
TEHQZ	Data Float After $\overline{ENABLE}$	0	48TCLCL	
TGHGL	$\overline{PROG}$ High to $\overline{PROG}$ Low	10		µs

EPROM Programming and Verification Waveforms

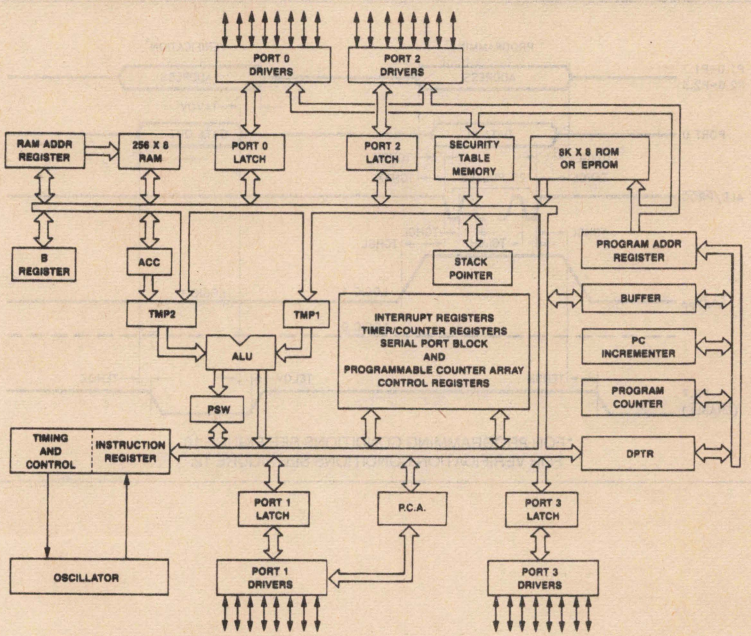


270147-15

*FOR PROGRAMMING CONDITIONS SEE FIGURE 10.
FOR VERIFICATION CONDITIONS SEE FIGURE 12.

CHMOS SINGLE-CHIP 8-BIT MICROCONTROLLER HSO, PWM, COMPARATOR/CAPTURE, UP/DOWN COUNTER 80C252—CPU WITH RAM AND I/O 83C252—8K BYTES FACTORY MASK PROGRAMMABLE ROM 87C252—8K BYTES USER PROGRAMMABLE EPROM

- High Performance CHMOS Process
- Power Control Modes
- Programmable Counter Array
- High Speed OUTPUT
- Pulse Width Modulator
- Up/Down Timer/Counter
- Watchdog Timer
- Two Level Program Lock System
- 8K Factory Mask ROM
- 256 Bytes of On-Chip Data RAM
- Quick-Pulse Programming™ Algorithm
- Boolean Processor
- 32 Programmable I/O Lines
- Three 16-Bit Timer/Counters
- 7 Interrupt Sources
- Programmable Serial Channel
- Framing Error Detection
- Automatic Address Recognition
- TTL Compatible Logic Levels
- 64K Program Memory Space
- 64K Data Memory Space
- MCS®-51 Instruction Set
- PLCC and DIP packagings



270146-1

Figure 1. Block Diagram

September 1986
Order Number: 270146-001

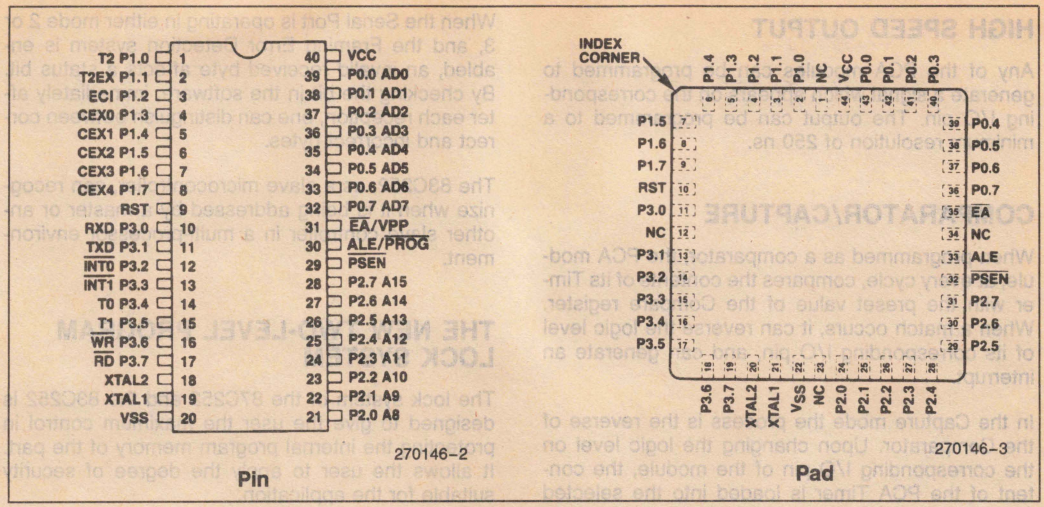


Figure 2. Pin Connection

The Intel 83C252 is a single-chip control oriented microcontroller which is fabricated on Intel's reliable CHMOS-III technology. Being a member of the MCS-51 family, the 83C252 uses the same powerful instruction set, has the same architecture, and is pin for pin backward compatible with the existing MCS-51 products.

The 83C252 has several new features that make it even more powerful than the 80C51BH. These features include: an additional 128 x 8 bytes of on-chip RAM, an extra Timer/Counter with up/down counting capability, a Programmable Counter Array, and Framing Error Detection and Automatic Address Recognition for the Serial Port.

The Programmable Counter Array consists of five modules and five I/O pins (share Port 1 pins). Each module is capable of being programmed as: a Pulse Width Modulator, a Compare/Capture Register, or a High Speed Output. Additionally, Module 4 can be programmed to be a Watch Dog Timer. The PCA can generate one interrupt when programmed as a Timer or as a Compare/Capture register.

With the Pulse Width Modulator and the High Speed Output capability, the 83C252 can be used in a wide range of motor control applications, and the Serial Port enhancements provide extra reliability in the multiprocessor serial communication environments.

The overall power consumption of this part, plus the two power control modes (Idle and Power Down), make this part ideal for low power and/or battery operated applications.

PULSE WIDTH MODULATOR

Any or all of the 5 modules of the PCA can be programmed to be a PWM. In this mode the PCA module can convert digital data to an analog signal. With a 16 MHz clock the maximum frequency of the output waveform of the PWM is 15.6 KHz. The frequency of the PWM depends on one of four software selectable clock sources of the PCA Timer. The duty cycle of the waveform is controlled by the contents of an 8-bit register that can be programmed to be any integer from 0 to 255.

HIGH SPEED OUTPUT

Any of the PCA modules can be programmed to generate a signal which appears on the corresponding I/O pin. The output can be programmed to a minimum resolution of 250 ns.

COMPARATOR/CAPTURE

When programmed as a comparator, the PCA module, at every cycle, compares the contents of its Timer with the preset value of the Compare register. When a match occurs, it can reverse the logic level of its corresponding I/O pin, and can generate an interrupt.

In the Capture mode the process is the reverse of the Comparator. Upon changing the logic level on the corresponding I/O pin of the module, the content of the PCA Timer is loaded into the selected Capture register. An interrupt may also be generated.

These two features allow the 83C252 to be used for accurate pulse width measurement in real time, with minimum software overhead.

WATCHDOG TIMER

Module number 4 of the PCA, along with its Timer can be programmed to play Watchdog Timer. If the Timer value matches the Comparator Register value, an internal Reset signal is generated which puts the microcontroller in a hardware reset.

During normal operation the software, periodically, reloads the Timer or disables the internal reset signal. Therefore, in case of a malfunction due to noise or any unwanted situation, the processor resets itself and provides more reliable operation.

TIMER/COUNTER

Timer 2 is a 16-bit timer/counter which is capable of up or down counting. It has a 16-bit register that can be used for auto-reloading or as a capture register. The capture command can come from external source as well as the clock input to the timer. With the help of Timer 2, two different baud rates can be generated for transmit and receive. Timer 2 can be utilized while the processor is in the Idle Mode.

SERIAL PORT

The full duplex serial port of the 83C252 is the same as the serial port of the 8052AH with two new features; Framing Error Detection, and Automatic Address Recognition.

When the Serial Port is operating in either mode 2 or 3, and the Framing Error Detection system is enabled, an invalid received byte affects a status bit. By checking the bit in the software, immediately after each reception, one can distinguish between correct and incorrect bytes.

The 83C252, as a slave microcontroller, can recognize when it is being addressed by a master or another slave controller in a multi-processor environment.

THE NEW TWO-LEVEL PROGRAM LOCK SYSTEM

The lock system of the 87C252 and the 83C252 is designed to give the user the maximum control in protecting the internal program memory of the part. It allows the user to apply the degree of security suitable for the application.

Two lock bits are implemented. Programming both bits denies any external access to the on-chip program memory.

The lock bits, when programmed, prohibit the controller from reading or moving the internal code when executing out of external program memory, and also disable the verify mode.

It is possible to maintain the verify capability while restricting access to the code. This is done by programming only one bit (SECURE EXTERNAL EXECUTION), and programming the SECURITY TABLE MEMORY. The data that appears on the port during the secured verification, is scrambled by the SECURITY TABLE contents.

In the 83C252 the Encryption Array is a 32 x 8 array of ROM. The user can electrically program any arbitrary code into it. The code will be used to mask the program memory during the verification. Thus what appears at port 0 during the verification is an encrypted code which cannot be deciphered without the key table.

In the 83C252 the Encryption Array and the Lock Bits are mask programmable.

Programming the lock bits also denies the programming of the EPROM and the Encryption Array. Therefore they must be locked after the code and the Encryption Array are programmed.

Erasing the EPROM also erases the Encryption Array and the lock bits. When erased the EPROM and the array contain all 1s and the part has its full functionality.

QUICK-PULSE PROGRAMMING™ ALGORITHM

The 87C252 can be programmed by using the Quick-Pulse Programming™ Algorithm for microcontrollers. The features of the new programming algorithm are a lower VPP (12.75V as compared with 21V) and a shorter programming pulse. It is possible to program the entire 8K Bytes of EPROM memory in less than 25 seconds with this algorithm!

MEMORY ORGANIZATION

Program Memory: Up to 8K bytes of the program memory can reside in the on-chip ROM. In addition the device can address up to 64K of program memory external to the chip.

Data Memory: This microcontroller has a 256 x 8 on-chip RAM. In addition it can address up to 64K bytes of external data memory.

PACKAGING INFORMATION

40 pin cerdip	D8 x C252
44 pin I_{CC}	R8 x C252

8031AH/8051AH
8032AH/8052AH
8751H

EXPRESS

■ **Extended Temperature Range** ■ **Burn-In**

The Intel EXPRESS system offers enhancements to the operational specifications of the MCS®-51 family of microcontrollers. These EXPRESS products are designed to meet the needs of those applications whose operating requirements exceed commercial standards.

The EXPRESS program includes the commercial standard temperature range with burn-in, and an extended temperature range with or without burn-in.

With the commercial standard temperature range operational characteristics are guaranteed over the temperature range of 0°C to 70°C. With the extended temperature range option, operational characteristics are guaranteed over the range of −40°C to +85°C.

The optional burn-in is dynamic, for a minimum time of 160 hours at 125°C with V_{CC} = 5.5V ±0.25V, following guidelines in MIL-STD-883, Method 1015.

Package types and EXPRESS versions are identified by a one- or two-letter prefix to the part number. The prefixes are listed in Table 1.

For the extended temperature range option, this data sheet specifies the parameters which deviate from their commercial temperature range limits. The commercial temperature range data sheets are applicable for all parameters not listed here.

Electrical Deviations from Commercial Specifications for Extended Temperature Range

D.C. and A.C. parameters not included here are the same as in the commercial temperature range data sheets.

D.C. CHARACTERISTICS T_A = −40°C to +85°C; V_{CC} = 5V ±10%; V_{SS} = 0V

Symbol	Parameter	Min	Max	Unit	Test Conditions
V_{IL}	Input Low Voltage	−0.5	0.75	V	
V_{IH}	Input High Voltage (Except XTAL2, RST)	2.1	V_{CC} + 0.5	V	
I_{CC}	Power Supply Current: 8051AH, 8031AH 8052AH, 8032AH 8751H, 8751H-8		 135 175 265	 mA mA mA	 All Outputs Disconnected; $\overline{EA}$ = V_{CC}
I_{IL2}	Logic 0 Input Current (XTAL2)		−4.0	mA	V_{in} = 0.45V

November 1986
Order Number: 270007-001

Table 1. Prefix Identification

Prefix	Package Type	Temperature Range	Burn-In
P	plastic	commercial	no
D	cerdip	commercial	no
C	ceramic	commercial	no
TP	plastic	extended	no
TD	cerdip	extended	no
TC	ceramic	extended	no
QP	plastic	commercial	yes
QD	cerdip	commercial	yes
QC	ceramic	commercial	yes
LP	plastic	extended	yes
LD	cerdip	extended	yes
LC	ceramic	extended	yes

Please note:

- Commercial temperature range is 0°C to 70°C. Extended temperature range is −40°C to +85°C.
- Burn-in is dynamic, for a minimum time of 160 hours at 125°C, V_{CC} = 5.5V ±0.25V, following guidelines in MIL-STD-883 Method 1015 (Test Condition D).
- The following device is not available in plastic packages:

  8751H
- The following devices are not available in ceramic packages:

  8051AH, 8031AH

  8052AH, 8032AH

Examples: P8031AH indicates 8031AH in a plastic package and specified for commercial temperature range, without burn-in. LD8751H indicates 8751H in a cerdip package and specified for extended temperature range with burn-in.

CHMOS SINGLE COMPONENT 8-BIT MICROCONTROLLER WITH FACTORY MASK-PROGRAMMABLE ROM

CHMOS SINGLE COMPONENT 8-BIT CONTROL-ORIENTED CPU WITH RAM AND I/O

80C51BH/80C31BH—3.5 to 12 MHz V_{CC} = 5V ± 10%

EXPRESS

- **Extended Temperature Range**
- **Burn-In**

The Intel EXPRESS system offers enhancements to the operational specifications of the MCS®-51 family of microcontrollers. These EXPRESS products are designed to meet the needs of those applications whose operating requirements exceed commercial standards.

The EXPRESS program includes the commercial standard temperature range with burn-in and an extended temperature range with or without burn-in.

With the commercial standard temperature range, operational characteristics are guaranteed over the temperature range of 0°C to 70°C. With the extended temperature range option, operational characteristics are guaranteed over the range of −40°C to +85°C.

The optional burn-in is dynamic for a minimum time of 160 hours at 125°C with V_{CC} = 6.0V ± 0.25V, following guidelines in MIL-STD-883, Method 1015.

Package types and EXPRESS versions are identified by a one- or two-letter prefix to the part number. The prefixes are listed in Table 1.

For the extended temperature range option, this data sheet specifies the parameters which deviate from their commercial temperature range limits. The commercial temperature range data sheets are applicable for all parameters not listed here.

October 1986
Order Number: 270218-001

Electrical Deviations from Commercial Specifications for Extended Temperature Range

D.C. and A.C. parameters not included here are the same as in the commercial temperature range data sheets.

D.C. CHARACTERISTICS $T_A = -40°C$ to $+85°C$; $V_{CC} = 5V \pm 10\%$; $V_{SS} = 0V$

Symbol	Parameter	Limits		Unit	Test Conditions
		Min	Max		
V_{IL}	Input Low Voltage (Except EA)	−0.5	$0.2V_{CC} - 0.15$	V	
V_{IL1}	EA	−0.5V	$0.2V_{CC} - 0.35$	V	
V_{IH}	Input High Voltage (Except XTAL1, RST)	$0.2V_{CC} + 1$	$V_{CC} + 0.5$	V	
V_{IH1}	Input High Voltage to XTAL1, RST	$0.7V_{CC} + 0.1$	$V_{CC} + 0.5$	V	
I_{IL}	Logical 0 Input Current (Port 1, 2, 3)	−75		μA	$V_{in} = 0.45V$
I_{TL}	Logical 1 to 0 transition Current (Ports 1, 2, 3)	−750		μA	$V_{in} = 2.0V$

Table 1. Prefix Identification

Prefix	Package Type	Temperature Range	Burn-In
P	Plastic	Commercial	No
D	Cerdip	Commercial	No
N	PLCC	Commercial	No
TP	Plastic	Extended	No
TD	Cerdip	Extended	No
TN	PLCC	Extended	No
QP	Plastic	Commercial	Yes
QD	Cerdip	Commercial	Yes
QN	PLCC	Commercial	Yes
LP	Plastic	Extended	Yes
LD	Cerdip	Extended	Yes
LN	PLCC	Extended	Yes

NOTE:
- Commercial temperature range is 0°C to 70°C. Extended temperature range is −40°C to +85°C.
- Burn-in is dynamic for a minimum time of 160 hours at 125°C, $V_{CC} = 6.0V \pm 0.25V$, following guidelines in MIL-STD-883 Method 1015 (Test Condition D).

Examples:

P80C31BH indicates 80C31BH in a plastic package and specified for commercial temperature range, without burn-in.
LD80C51BH indicates 80C51BH in a cerdip package and specified for extended temperature range with burn-in.

Electrical Deviations from Commercial Specifications for Extended Temperature Range

D.C. and A.C. parameters not included here are the same as in the commercial temperature range data sheets.

D.C. CHARACTERISTICS T_A = −40°C to +85°C; V_{CC} = 5V ± 10%; V_{SS} = 0V

Symbol	Parameter	Limits		Unit	Test Conditions
		Min	Max		
V_{IL}	Input Low Voltage (Except EA)	−0.5	0.2V_{CC} − 0.15	V	
V_{IL1}	EA	−0.5V	0.2V_{CC} − 0.35	V	
V_{IH}	Input High Voltage (Except XTAL1, RST)	0.2V_{CC} + .7	V_{CC} + 0.5	V	
V_{IH1}	Input High Voltage to XTAL1, RST	0.7V_{CC} + 0.1	V_{CC} + 0.5	V	
I_{IL}	Logical 0 Input Current (Port 1, 2, 3)		−75	µA	V_{IN} = 0.45V
I_{TL}	Logical 1 to 0 transition Current (Ports 1, 2, 3)		−750	µA	V_{IN} = 2.0V

Table 1. Prefix Identification

Prefix	Package Type	Temperature Range	Burn-In
P	Plastic	Commercial	No
D	Cerdip	Commercial	No
N	PLCC	Commercial	No
TP	Plastic	Extended	No
TD	Cerdip	Extended	No
TN	PLCC	Extended	No
QP	Plastic	Commercial	Yes
QD	Cerdip	Commercial	Yes
QN	PLCC	Commercial	Yes
LP	Plastic	Extended	Yes
LD	Cerdip	Extended	Yes
LN	PLCC	Extended	Yes

NOTE:
• Commercial temperature range is 0°C to 70°C. Extended temperature range is −40°C to +85°C.
• Burn-in is dynamic for a minimum time of 160 hours at 125°C, V_{CC} = 5.0V ± 0.25V, following guidelines in MIL-STD-883 Method 1015 (Test Condition D).

Examples:

P80C31BH indicates 80C31BH in a plastic package and specified for commercial temperature range, without burn-in.

LD80C51BH indicates 80C51BH in a cerdip package and specified for extended temperature range with burn-in.

MCS®-51 Application Notes 9

September 1986

Using the Intel MCS®-51 Boolean Processing Capabilities

JOHN WHARTON
MICROCONTROLLER APPLICATIONS

Order Number: 203830-001

1.0 INTRODUCTION

The Intel microcontroller family now has three new members: the Intel® 8031, 8051, and 8751 single-chip microcomputers. These devices, shown in Figure 1, will allow whole new classes of products to benefit from recent advances in Integrated Electronics. Thanks to Intel's new HMOS technology, they provide larger program and data memory spaces, more flexible I/O and peripheral capabilities, greater speed, and lower system cost than any previous-generation single-chip microcomputer.

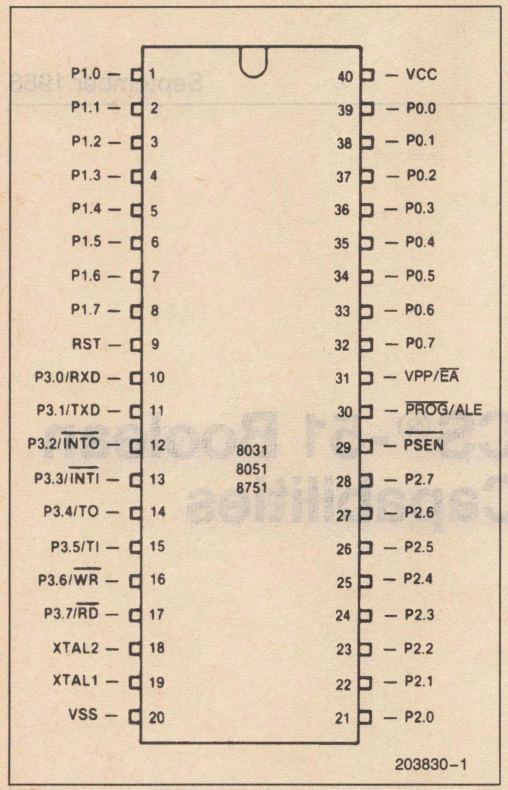


Figure 1. 8051 Family Pinout Diagram

Table 1 summarizes the quantitative differences between the members of the MCS®-48 and 8051 families. The 8751 contains 4K bytes of EPROM program memory fabricated on-chip, while the 8051 replaces the EPROM with 4K bytes of lower-cost mask-programmed ROM. The 8031 has no program memory on-chip; instead, it accesses up to 64K bytes of program memory from external memory. Otherwise, the three new family members are identical. Throughout this Note, the term "8051" will represent all members of the 8051 Family, unless specifically stated otherwise.

The CPU in each microcomputer is one of the industry's fastest and most efficient for numerical calculations on byte operands. But controllers often deal with bits, not bytes: in the real world, switch contacts can only be open or closed, indicators should be either lit or dark, motors are either turned on or off, and so forth. For such control situations the most significant aspect of the MCS®-51 architecture is its complete hardware support for one-bit, or *Boolean* variables (named in honor of Mathematician George Boole) as a separate data type.

The 8051 incorporates a number of special features which support the direct manipulation and testing of individual bits and allow the use of single-bit variables in performing logical operations. Taken together, these features are referred to as the MCS-51 *Boolean Processor*. While the bit-processing capabilities alone would be adequate to solve many control applications, their true power comes when they are used in conjunction with the microcomputer's byte-processing and numerical capabilities.

Many concepts embodied by the Boolean Processor will certainly be new even to experienced microcomputer system designers. The purpose of this Application Note is to explain these concepts and show how they are used.

For detailed information on these parts refer to the **Intel Microcontroller Handbook,** order number 210918. The instruction set, assembly language, and use of the 8051 assembler (ASM51) are further described in the **MCS®-51 Macro Assembler User's Guide for DOS Systems,** order number 122753.

Table 1. Features of Intel's Single-Chip Microcomputers

EPROM Program Memory	ROM Program Memory	External Program Memory	Program Memory (Int/Max)	Data Memory (Bytes)	Instr. Cycle Time	Input/ Output Pins	Interrupt Sources	Reg. Banks
8748	8048	8035	1K 4K	64	2.5 μs	27	2	2
—	8049	8039	2K 4K	128	1.36 μs	27	2	2
8751	8051	8031	4K 64K	128	1.0 μs	32	5	4

2.0 BOOLEAN PROCESSOR OPERATION

The Boolean Processing capabilities of the 8051 are based on concepts which have been around for some time. Digital computer systems of widely varying designs all have four functional elements in common (Figure 2):

- a central processor (CPU) with the control, timing, and logic circuits needed to execute stored instructions:
- a memory to store the sequence of instructions making up a program or algorithm:
- data memory to store variables used by the program:
  and
- some means of communicating with the outside world.

The CPU usually includes one or more accumulators or special registers for computing or storing values during program execution. The instruction set of such a processor generally includes, at a minimum, operation classes to perform arithmetic or logical functions on program variables, move variables from one place to another, cause program execution to jump or conditionally branch based on register or variable states, and instructions to call and return from subroutines. The program and data memory functions sometimes share a single memory space, but this is not always the case. When the address spaces are separated, program and data memory need not even have the same basic word width.

A digital computer's flexibility comes in part from combining simple fast operations to produce more complex (albeit slower) ones, which in turn link together eventually solving the problem at hand. A four-bit CPU executing multiple precision subroutines can, for example, perform 64-bit addition and subtraction. The subroutines could in turn be building blocks for floating-point multiplication and division routines. Eventually, the four-bit CPU can simulate a far more complex "virtual" machine.

In fact, *any* digital computer with the above four functional elements can (given time) complete *any* algorithm (though the proverbial room full of chimpanzees at word processors might first re-create Shakespeare's classics and this Application Note)! This fact offers little consolation to product designers who want programs to run as quickly as possible. By definition, a real-time control algorithm *must* proceed quickly enough to meet the preordained speed constraints of other equipment.

One of the factors determining how long it will take a microcomputer to complete a given chore is the number of instructions it must execute. What makes a given computer architecture particularly well- or poorly-suited for a class of problems is how well its instruction set matches the tasks to be performed. The better the "primitive" operations correspond to the steps taken by the control algorithm, the lower the number of instructions needed, and the quicker the program will run. All else being equal, a CPU supporting 64-bit arithmetic directly could clearly perform floating-point math faster than a machine bogged-down by multiple-precision subroutines. In the same way, direct support for bit manipulation naturally leads to more efficient programs handling the binary input and output conditions inherent in digital control problems.

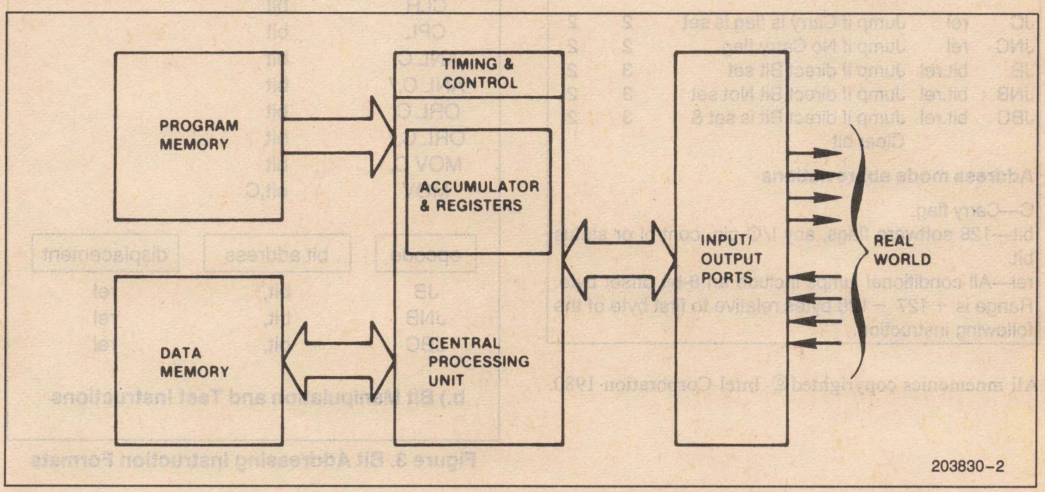


Figure 2. Block Diagram for Abstract Digital Computer

Processing Elements

The introduction stated that the 8051's bit-handling capabilities alone would be sufficient to solve some control applications. Let's see how the four basic elements of a digital computer—a CPU with associated registers, program memory, addressable data RAM, and I/O capability—relate to Boolean variables.

CPU. The 8051 CPU incorporates special logic devoted to executing several bit-wide operations. All told, there are 17 such instructions, all listed in Table 2. Not shown are 94 other (mostly byte-oriented) 8051 instructions.

Program Memory. Bit-processing instructions are fetched from the same program memory as other arithmetic and logical operations. In addition to the instructions of Table 2, several sophisticated program control features like multiple addressing modes, subroutine nesting, and a two-level interrupt structure are useful in structuring Boolean Processor-based programs.

Table 2. MCS-51™ Boolean Processing Instruction Subset

Mnemonic		Description	Byte	Cyc
SETB	C	Set Carry flag	1	1
SETB	bit	Set direct Bit	2	1
CLR	C	Clear Carry flag	1	1
CLR	bit	Clear direct bit	2	1
CPL	C	Complement Carry flag	1	1
CPL	bit	Complement direct bit	2	1
MOV	C.bit	Move direct bit to Carry flag	2	1
MOV	bit.C	Move Carry flag to direct bit	2	2
ANL	C.bit	AND direct bit to Carry flag	2	2
ANL	C.bit	AND complement of direct bit to Carry flag	2	2
ORL	C.bit	OR direct bit to Carry flag	2	2
ORL	C.bit	OR complement of direct bit to Carry flag	2	2
JC	rel	Jump if Carry is flag is set	2	2
JNC	rel	Jump if No Carry flag	2	2
JB	bit.rel	Jump if direct Bit set	3	2
JNB	bit.rel	Jump if direct Bit Not set	3	2
JBC	bit.rel	Jump if direct Bit is set & Clear bit	3	2

Address mode abbreviations

C—Carry flag.
bit—128 software flags, any I/O pin, control or status bit.
rel—All conditional jumps include an 8-bit offset byte. Range is +127 −128 bytes relative to first byte of the following instruction.

All mnemonics copyrighted© Intel Corporation 1980.

Boolean instructions are one, two, or three bytes long, depending on what function they perform. Those involving only the carry flag have either a single-byte opcode or an opcode followed by a conditional-branch destination byte (Figure 3a). The more general instructions add a "direct address" byte after the opcode to specify the bit affected, yielding two or three byte encodings (Figure 3b). Though this format allows potentially 256 directly addressable bit locations, not all of them are implemented in the 8051 family.

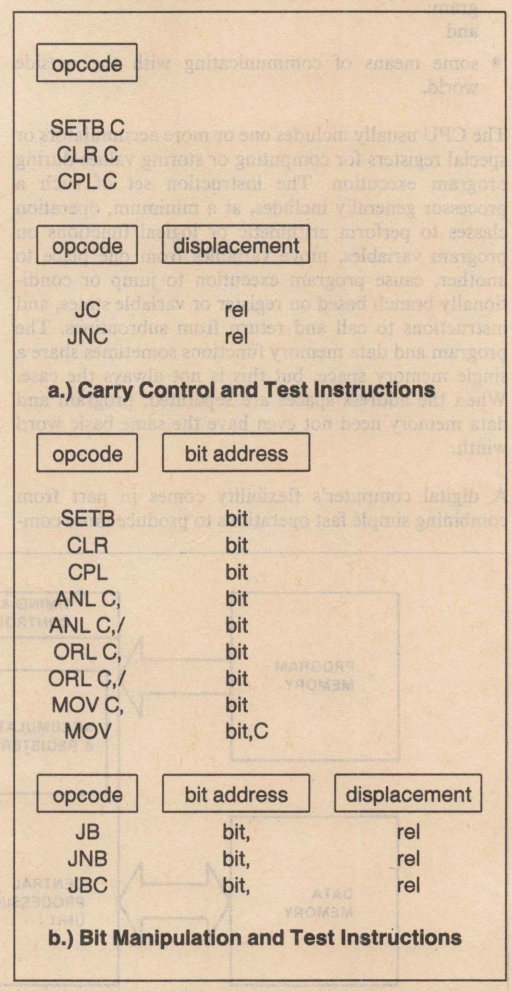


Figure 3. Bit Addressing Instruction Formats

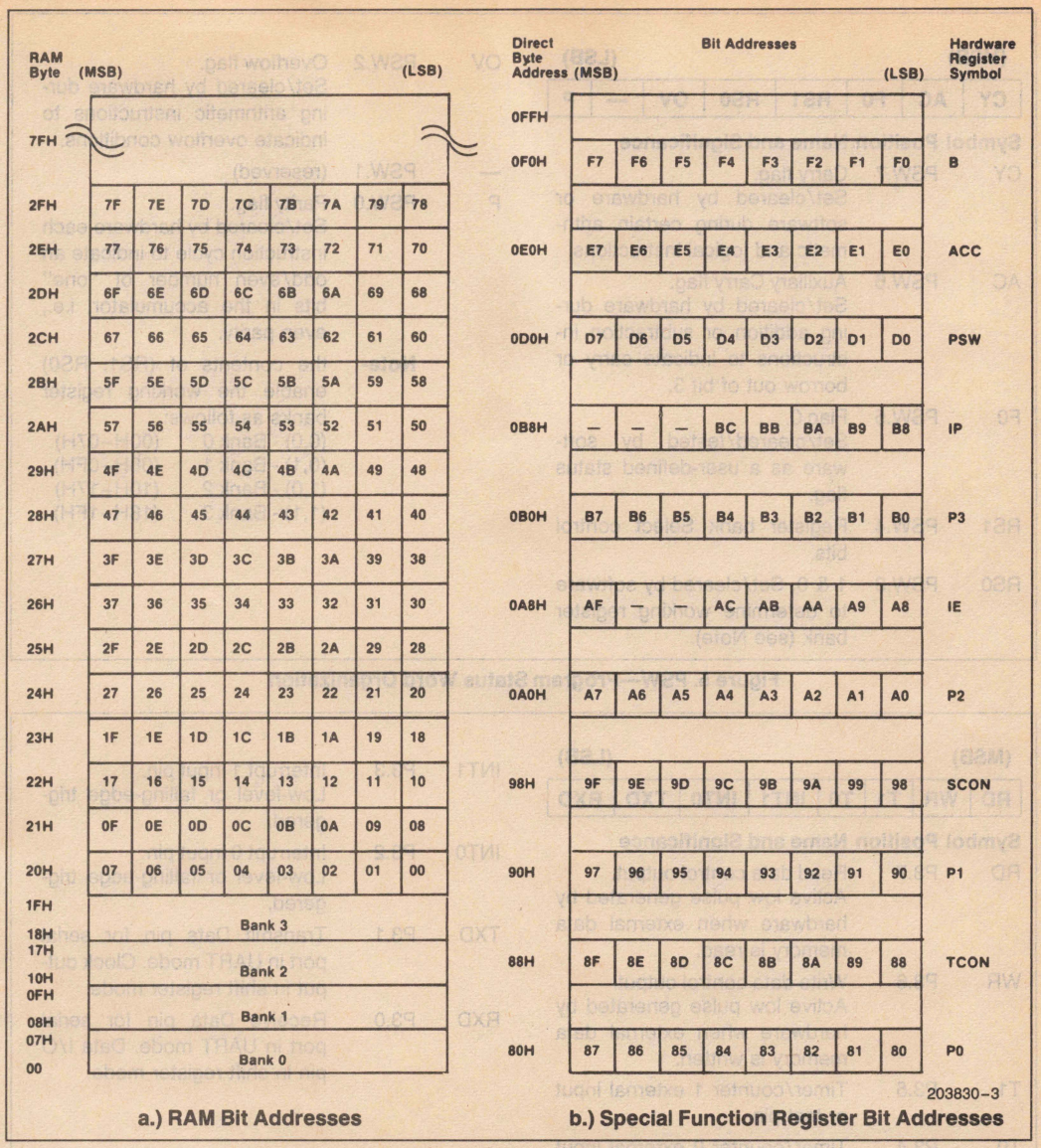


Figure 4. Bit Address Maps

Data Memory. The instructions in Figure 3b can operate directly upon 144 general purpose bits forming the Boolean processor "RAM." These bits can be used as software flags or to store program variables. Two operand instructions use the CPU's carry flag ("C") as a special one-bit register: in a sense, the carry is a "Boolean accumulator" for logical operations and data transfers.

Input/Output. All 32 I/O pins can be addressed as individual inputs, outputs, or both, in any combination. Any pin can be a control strobe output, status (Test) input, or serial I/O link implemented via software. An additional 33 individually addressable bits reconfigure, control, and monitor the status of the CPU and all on-chip peripheral functions (timer counters, serial port modes, interrupt logic, and so forth).

(MSB)							(LSB)
CY	AC	F0	RS1	RS0	OV	—	P

Symbol	Position	Name and Significance
CY	PSW.7	Carry flag. Set/cleared by hardware or software during certain arithmetic and logical instructions.
AC	PSW.6	Auxiliary Carry flag. Set/cleared by hardware during addition or subtraction instructions to indicate carry or borrow out of bit 3.
F0	PSW.5	Flag 0. Set/cleared/tested by software as a user-defined status flag.
RS1	PSW.4	Register bank Select control bits.
RS0	PSW.3	1 & 0. Set/cleared by software to determine working register bank (see Note).
OV	PSW.2	Overflow flag. Set/cleared by hardware during arithmetic instructions to indicate overflow conditions.
—	PSW.1	(reserved)
P	PSW.0	Parity flag. Set/cleared by hardware each instruction cycle to indicate an odd/even number of "one" bits in the accumulator, i.e., even parity.

Note- the contents of (RS1, RS0) enable the working register banks as follows:

(0,0) - Bank 0	(00H–07H)
(0,1) - Bank 1	(08H–0FH)
(1,0) - Bank 2	(10H–17H)
(1,1) - Bank 3	(18H–1FH)

Figure 5. PSW—Program Status Word Organization

(MSB)							(LSB)
RD	WR	T1	T0	INT1	INT0	TXD	RXD

Symbol	Position	Name and Significance
RD	P3.7	Read data control output. Active low pulse generated by hardware when external data memory is read.
WR	P3.6	Write data control output. Active low pulse generated by hardware when external data memory is written.
T1	P3.5	Timer/counter 1 external input or test pin.
T0	P3.4	Timer/counter 0 external input or test pin.
INT1	P3.3	Interrupt 1 input pin. Low-level or falling-edge triggered.
INT0	P3.2	Interrupt 0 input pin. Low-level or falling-edge triggered.
TXD	P3.1	Transmit Data pin for serial port in UART mode. Clock output in shift register mode.
RXD	P3.0	Receive Data pin for serial port in UART mode. Data I/O pin in shift register mode.

Figure 6. P3—Alternate I/O Functions of Port 3

Direct Bit Addressing

The most significant bit of the direct address byte selects one of two groups of bits. Values between 0 and 127 (00H and 7FH) define bits in a block of 32 bytes of on-chip RAM, between RAM addresses 20H and 2FH (Figure 4a). They are numbered consecutively from the lowest-order byte's lowest-order bit through the highest-order byte's highest-order bit.

Bit addresses between 128 and 255 (80H and 0FFH) correspond to bits in a number of special registers, mostly used for I/O or peripheral control. These positions are numbered with a different scheme than RAM: the five high-order address bits match those of the register's own address, while the three low-order bits identify the bit position within that register (Figure 4b).

Notice the column labeled "Symbol" in Figure 5. Bits with special meanings in the PSW and other registers have corresponding symbolic names. General-purpose (as opposed to carry-specific) instructions may access the carry like any other bit by using the mnemonic CY in place of C, P0, P1, P2, and P3 are the 8051's four I/O ports: secondary functions assigned to each of the eight pins of P3 are shown in Figure 6.

Figure 7 shows the last four bit addressable registers. TCON (Timer Control) and SCON (Serial port Control) control and monitor the corresponding peripherals, while IE (Interrupt Enable) and IP (Interrupt Priority) enable and prioritize the five hardware interrupt sources. Like the reserved hardware register addresses, the five bits not implemented in IE and IP should not be accessed: they can *not* be used as software flags.

Addressable Register Set. There are 20 special function registers in the 8051, but the advantages of bit addressing only relate to the 11 described below. Five potentially bit-addressable register addresses (0C0H, 0C8H, 0D8H, 0E8H, & 0F8H) are being reserved for possible future expansion in microcomputers based on the MCS-51 architecture. Reading or writing non-existent registers in the 8051 series is pointless, and may cause unpredictable results. Byte-wide logical operations can be used to manipulate bits in all *non*-bit addressable registers and RAM.

(MSB)							(LSB)
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

Symbol Position Name and Significance

TF1 TCON.7 Timer 1 overflow Flag. Set by hardware on timer/counter overflow. Cleared when interrupt processed.

TR1 TCON.6 Timer 1 Run control bit. Set/cleared by software to turn timer/counter on/off.

TF0 TCON.5 Timer 0 overflow Flag. Set by hardware on timer/counter overflow. Cleared when interrupt processed.

TR0 TCON.4 Timer 0 Run control bit. Set/cleared by software to turn timer/counter on/off.

IE1 TCON.3 Interrupt 1 Edge flag. Set by hardware when external interrupt edge detected. Cleared when interrupt processed.

IT1 TCON.2 Interrupt 1 Type control bit. Set/cleared by software to specify falling edge/low level triggered external interrupts.

IE0 TCON.1 Interrupt 0 Edge flag. Set by hardware when external interrupt edge detected. Cleared when interrupt processed.

IT0 TCON.0 Interrupt 0 Type control bit. Set/cleared by software to specify falling edge/low level triggered external interrupts.

a.) TCON—Timer/Counter Control/Status Register

(MSB)							(LSB)
SM0	SM1	SM2	REN	TB8	RB8	TI	RI

Symbol Position Name and Significance

SM0 SCON.7 Serial port Mode control bit 0. Set/cleared by software (see note).

SM1 SCON.6 Serial port Mode control bit 1. Set/cleared by software (see note).

SM2 SCON.5 Serial port Mode control bit 2. Set by software to disable reception of frames for which bit 8 is zero.

REN SCON.4 Receiver Enable control bit. Set/cleared by software to enable/disable serial data reception.

TB8 SCON.3 Transmit Bit 8. Set/cleared by hardware to determine state of ninth data bit transmitted in 9-bit UART mode.

RB8 SCON.2 Receive Bit 8. Set/cleared by hardware to indicate state of ninth data bit received.

TI SCON.1 Transmit Interrupt flag. Set by hardware when byte transmitted. Cleared by software after servicing.

RI SCON.0 Receive Interrupt flag. Set by hardware when byte received. Cleared by software after servicing.

Note- the state of (SM0, SM1) selects:
(0,0)—Shift register I/O expansion.
(0,1)—8-bit UART, variable data rate.
(1,0)—9-bit UART, fixed data rate.
(1,1)—9-bit UART, variable data rate.

b.) SCON—Serial Port Control/Status Register

Figure 7. Peripheral Configuration Registers

(MSB)							(LSB)
EA	—	—	ES	ET1	EX1	ET1	EX0

Symbol	Position	Name and Significance
EA	IE.7	Enable All control bit. Cleared by software to disable all interrupts, independent of the state of IE.4–IE.0.
—	IE.6	(reserved)
—	IE.5	
ES	IE.4	Enable Serial port control bit. Set/cleared by software to enable/disable interrupts from TI or RI flags.
ET1	IE.3	Enable Timer 1 control bit. Set/cleared by software to enable/disable interrupts from timer/counter 1.
EX1	IE.2	Enable External interrupt 1 control bit. Set/cleared by software to enable/disable interrupts from INT1.
ET0	IE.1	Enable Timer 0 control bit. Set/cleared by software to enable/disable interrupts from timer/counter 0.
EX0	IE.0	Enable External interrupt 0 control bit. Set/cleared by software to enable/disable interrupts from INT0.

c.) IE—Interrupt Enable Register

(MSB)							(LSB)
—	—	—	PS	PT1	PX1	PT0	PX0

Symbol	Position	Name and Significance
—	IP.7	(reserved)
—	IP.6	(reserved)
—	IP.5	(reserved)
PS	IP.4	Serial port Priority control bit. Set/cleared by software to specify high/low priority interrupts for Serial port.
PT1	IP.3	Timer 1 Priority control bit. Set/cleared by software to specify high/low priority interrupts for timer/counter 1.
PX1	IP.2	External interrupt 1 Priority control bit. Set/cleared by software to specify high/low priority interrupts for INT1.
PT0	IP.1	Timer 0 Priority control bit. Set/cleared by software to specify high/low priority interrupts for timer/counter 0.
PX0	IP.0	External interrupt 0 Priority control bit. Set/cleared by software to specify high/low priority interrupts for INT0.

d.) IP—Interrupt Priority Control Register

Figure 7. Peripheral Configuration Registers (Continued)

The accumulator and B registers (A and B) are normally involved in byte-wide arithmetic, but their individual bits can also be used as 16 general software flags. Added with the 128 flags in RAM, this gives 144 general purpose variables for bit-intensive programs. The program status word (PSW) in Figure 5 is a collection of flags and machine status bits including the carry flag itself. Byte operations acting on the PSW can therefore affect the carry.

Instruction Set

Having looked at the bit variables available to the Boolean Processor, we will now look at the four classes of instructions that manipulate these bits. It may be helpful to refer back to Table 2 while reading this section.

State Control. Addressable bits or flags may be set, cleared, or logically complemented in one instruction cycle with the two-byte instructions SETB, CLR, and CPL. (The "B" affixed to SETB distinguishes it from the assembler "SET" directive used for symbol definition.) SETB and CLR are analogous to loading a bit with a constant: 1 or 0. Single byte versions perform the same three operations on the carry.

The MCS-51 assembly language specifies a bit address in any of three ways:

- by a number or expression corresponding to the direct bit address (0–255):

- by the name or address of the register containing the bit, the *dot operator* symbol (a period: "."), and the bit's position in the register (7–0):
- in the case of control and status registers, by the predefined assembler symbols listed in the first columns of Figures 5–7.

Bits may also be given user-defined names with the assembler "BIT" directive and any of the above techniques. For example, bit 5 of the PSW may be cleared by any of the four instructions.

```
USR_FLG BIT   PSW.5     ; User Symbol Definition
        ...   ...
        CLR   0D5H      ; Absolute Addressing
        CLR   PSW.5     ; Use of Dot Operator
        CLR   F0        ; Pre-Defined Assembler
                        ; Symbol
        CLR   USR_FLG   ; User-Defined Symbol
```

Data Transfers. The two-byte MOV instructions can transport any addressable bit to the carry in one cycle, or copy the carry to the bit in two cycles. A bit can be moved between two arbitrary locations via the carry by combining the two instructions. (If necessary, push and pop the PSW to preserve the previous contents of the carry.) These instructions can replace the multi-instruction sequence of Figure 8, a program structure appearing in controller applications whenever flags or outputs are conditionally switched on or off.

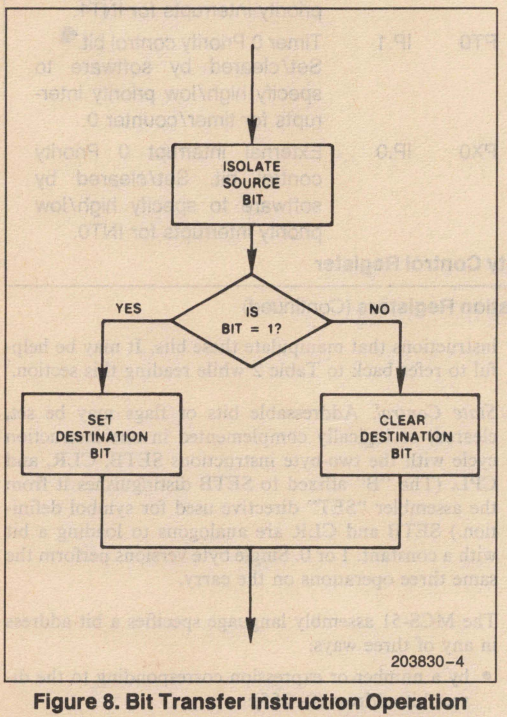


Figure 8. Bit Transfer Instruction Operation

Logical Operations. Four instructions perform the logical-AND and logical-OR operations between the carry and another bit, and leave the results in the carry. The instruction mnemonics are ANL and ORL; the absence or presence of a slash mark ("/") before the source operand indicates whether to use the positive-logic value or the logical complement of the addressed bit. (The source operand itself is never affected.)

Bit-test Instructions. The conditional jump instructions "JC rel" (Jump on Carry) and "JNC rel" (Jump on Not Carry) test the state of the carry flag, branching if it is a one or zero, respectively. (The letters "rel" denote relative code addressing.) The three-byte instructions "JB bit.rel" and "JNB bit.rel" (Jump on Bit and Jump on Not Bit) test the state of *any* addressable bit in a similar manner. A fifth instruction combines the Jump on Bit and Clear operations. "JBC bit.rel" conditionally branches to the indicated address, then clears the bit in the same two cycle instruction. This operation is the same as the MCS-48 "JTF" instructions.

All 8051 conditional jump instructions use program counter-relative addressing, and all execute in two cycles. The last instruction byte encodes a signed displacement ranging from −128 to +127. During execution, the CPU adds this value to the incremented program counter to produce the jump destination. Put another way, a conditional jump to the immediately following instruction would encode 00H in the offset byte.

A section of program or subroutine written using only relative jumps to nearby addresses will have the same machine code independent of the code's location. An assembled routine may be repositioned anywhere in memory, even crossing memory page boundaries, without having to modify the program or recompute destination addresses. To facilitate this flexibility, there is an unconditional "Short Jump" (SJMP) which uses relative addressing as well. Since a programmer would have quite a chore trying to compute relative offset values from one instruction to another, ASM51 automatically computes the displacement needed given only the destination address or label. An error message will alert the programmer if the destination is "out of range."

The so-called "Bit Test" instructions implemented on many other microprocessors simply perform the logical-AND operation between a byte variable and a constant mask, and set or clear a zero flag depending on the result. This is essentially equivalent to the 8051 "MOV C,bit" instruction. A second instruction is then needed to conditionally branch based on the state of the zero flag. This does *not* constitute abstract bit-addressing in the MCS-51 sense. A flag exists only as a field

within a register: to reference a bit the programmer must know and specify both the encompassing register and the bit's position therein. This constraint severely limits the flexibility of symbolic bit addressing and reduces the machine's code-efficiency and speed.

Interaction with Other Instructions. The carry flag is also affected by the instructions listed in Table 3. It can be rotated through the accumulator, and altered as a side effect of arithmetic instructions. Refer to the User's Manual for details on how these instructions operate.

Simple Instruction Combinations

By combining general purpose bit operations with certain addressable bits, one can "custom build" several hundred useful instructions. All eight bits of the PSW can be tested directly with conditional jump instructions to monitor (among other things) parity and overflow status. Programmers can take advantage of 128 software flags to keep track of operating modes, resource usage, and so forth.

The Boolean instructions are also the most efficient way to control or reconfigure peripheral and I/O registers. All 32 I/O lines become "test pins," for example, tested by conditional jump instructions. Any output pin can be toggled (complemented) in a single instruction cycle. Setting or clearing the Timer Run flags (TR0 and TR1) turn the timer/counters on or off; polling the same flags elsewhere lets the program determine if a timer is running. The respective overflow flags (TF0 and TF1) can be tested to determine when the desired period or count has elapsed, then cleared in preparation for the next repetition. (For the record, these bits are all part of the TCON register, Figure 7a. Thanks to symbolic bit addressing, the programmer only needs to remember the mnemonic associated with each function. In other words, don't bother memorizing control word layouts.)

In the MCS-48 family, instructions corresponding to some of the above functions require specific opcodes. Ten different opcodes serve to clear complement the software flags F0 and F1, enable/disable each interrupt, and start/stop the timer. In the 8051 instruction set, just three opcodes (SETB, CLR, CPL) with a direct bit address appended perform the same functions. Two test instructions (JB and JNB) can be combined with bit addresses to test the software flags, the 8048 I/O pins T0, T1, and INT, and the eight accumulator bits, replacing 15 more different instructions.

Table 4a shows how 8051 programs implement software flag and machine control functions associated with special opcodes in the 8048. In every case the MCS-51 solution requires the same number of machine cycles, and executes 2.5 times faster.

Table 3. Other Instructions Affecting the Carry Flag

Mnemonic		Description	Byte	Cyc
ADD	A,Rn	Add register to Accumulator	1	1
ADD	A,direct	Add direct byte to Accumulator	2	1
ADD	A,@Ri	Add indirect RAM to Accumulator	1	1
ADD	A,#data	Add immediate data to Accumulator	2	1
ADDC	A,Rn	Add register to Accumulator with Carry flag	1	1
ADDC	A,direct	Add direct byte to Accumulator with Carry flag	2	1
ADDC	A,@Ri	Add indirect RAM to Accumulator with Carry flag	1	1
ADDC	A,#data	Add immediate data to Acc with Carry flag	2	1
SUBB	A,Rn	Subtract register from Accumulator with borrow	1	1
SUBB	A,direct	Subtract direct byte from Acc with borrow	2	1
SUBB	A,@Ri	Subtract indirect RAM from Acc with borrow	1	1
SUBB	A,#data	Subtract immediate data from Acc with borrow	2	1
MUL	AB	Multiply A & B	1	4
DIV	AB	Divide A by B	1	4
DA	A	Decimal Adjust Accumulator	1	1
RLC	A	Rotate Accumulator Left through the Carry flag	1	1
RRC	A	Rotate Accumulator Right through Carry flag	1	1
CJNE	A,direct.rel	Compare direct byte to Acc & Jump if Not Equal	3	2
CJNE	A,#data.rel	Compare immediate to Acc & Jump if Not Equal	3	2
CJNE	Rn,#data.rel	Compare immed to register & Jump if Not Equal	3	2
CJNE	@Ri,#data.rel	Compare immed to indirect & Jump if Not Equal	3	2

All mnemonics copyrighted © Intel Corporation 1980.

Table 4a. Contrasting 8048 and 8051 Bit Control and Testing Instructions

8048 Instruction		Bytes	Cycles	µSec	8x51 Instruction		Bytes	Cycles & µSec
Flag Control								
CLR	C	1	1	2.5	CLR	C	1	1
CPL	F0	1	1	2.5	CPL	F0	2	1
Flag Testing								
JNC	offset	2	2	5.0	JNC	rel	2	2
JF0	offset	2	2	5.0	JB	F0.rel	3	2
JB7	offset	2	2	5.0	JB	ACC.7.rel	3	2
Peripheral Polling								
JT0	offset	2	2	5.0	JB	T0.rel	3	2
JN1	offset	2	2	5.0	JNB	INT0.rel	3	2
JTF	offset	2	2	5.0	JBC	TF0.rel	3	2
Machine and Peripheral Control								
STRT	T	1	1	2.5	SETB	TR0	2	1
EN	I	1	1	2.5	SETB	EX0	2	1
DIS	TCNT1	1	1	2.5	CLR	ET0	2	1

Table 4b. Replacing 8048 Instruction Sequences with Single 8x51 Instructions

8048 Instruction		Bytes	Cycles	µSec	8051 Instruction		Bytes	Cycles & µSec
Flag Control								
Set carry								
CLR	C							
CPL	C	= 2	2	5.0	SETB	C	1	1
Set Software Flag								
CLR	F0							
CPL	F0	= 2	2	5.0	SETB	F0	2	1
Turn Off Output Pin								
ANL	P1.#0FBH	= 2	2	5.0	CLR	P1.2	2	1
Complement Output Pin								
IN	A.P1							
XRL	A.#04H							
OUTL	P1.A	= 4	6	15.0	CPL	P1.2	2	1
Clear Flag in RAM								
MOV	R0.#FLGADR							
MOV	A.@R0							
ANL	A.#FLGMASK							
MOV	@R0.A	= 6	6	15.0	CLR	USER_FLG	2	1

Table 4b. Replacing 8048 Instruction Sequences with Single 8x51 Instructions (Continued)

8048 Instruction	Bytes	Cycles	μSec	8x51 Instruction	Bytes	Cycles & μSec
Flag Testing:						
Jump if Software Flag is 0						
JF0 $+4						
JMP offset	= 4	4	10.0	JNB F0.rel	3	2
Jump if Accumulator bit is 0						
CPL A						
JB7 offset						
CPL A	= 4	4	10.0	JNB ACC.7.rel	3	2
Peripheral Polling						
Test if Input Pin is Grounded						
IN A.P1						
CPL A						
JB3 offset	= 4	5	12.5	JNB P1.3.rel	3	2
Test if Interrupt Pin is High						
JN1 $+4						
JMP offset	= 4	4	10.0	JB INT0.rel	3	2

3.0 BOOLEAN PROCESSOR APPLICATIONS

So what? Then what does all this buy you?

Qualitatively, nothing. All the same capabilities *could* be (and often have been) implemented on other machines using awkward sequences of other basic operations. As mentioned earlier, any CPU can solve any problem given enough time.

Quantitatively, the differences between a solution allowed by the 8051 and those required by previous architectures are numerous. What the 8051 Family buys you is a faster, cleaner, lower-cost solution to microcontroller applications.

The opcode space freed by condensing many specific 8048 instructions into a few general operations has been used to add new functionality to the MCS-51 architecture—both for byte and bit operations. 144 software flags replace the 8048's two. These flags (and the carry) may be directly set, not just cleared and complemented, and all can be tested for either state, not just one. Operating mode bits previously inaccessible may be read, tested, or saved. Situations where the 8051 instruction set provides new capabilities are contrasted with 8048 instruction sequences in Table 4b. Here the 8051 speed advantage ranges from 5x to 15x!

Combining Boolean and byte-wide instructions can produce great synergy. An MCS-51 based application will prove to be:

- simpler to write since the architecture correlates more closely with the problems being solved:
- easier to debug because more individual instructions have no unexpected or undesirable side-effects:
- more byte efficient due to direct bit addressing and program counter relative branching:
- faster running because fewer bytes of instruction need to be fetched and fewer conditional jumps are processed:
- lower cost because of the high level of system-integration within one component.

These rather unabashed claims of excellence shall not go unsubstantiated. The rest of this chapter examines less trivial tasks simplified by the Boolean processor. The first three compare the 8051 with other microprocessors; the last two go into 8051-based system designs in much greater depth.

Design Example # 1—Bit Permutation

First off, we'll use the bit-transfer instructions to permute a lengthy pattern of bits.

A steadily increasing number of data communication products use encoding methods to protect the security of sensitive information. By law, interstate financial transactions involving the Federal banking system must be transmitted using the Federal Information Processing *Data Encryption Standard* (DES).

Basically, the DES combines eight bytes of "plaintext" data (in binary, ASCII, or any other format) with a 56-bit "key", producing a 64-bit encrypted value for transmission. At the receiving end the same algorithm is applied to the incoming data using the same key, reproducing the original eight byte message. The algorithm used for these permutations is fixed; different user-defined keys ensure data privacy.

It is not the purpose of this note to describe the DES in any detail. Suffice it to say that encryption/decryption is a long, iterative process consisting of rotations, exclusive -OR operations, function table look-ups, and an extensive (and quite bizarre) sequence of bit permutation, packing, and unpacking steps. (For further details refer to the June 21, 1979 issue of **Electronics** magazine.) The bit manipulation steps are included, it is rumored, to impede a general purpose digital supercomputer trying to "break" the code. Any algorithm implementing the DES with previous generation microprocessors would spend virtually all of its time diddling bits.

The bit manipulation performed is typified by the Key Schedule Calculation represented in Figure 9. This step is repeated 16 times for each key used in the course of a transmission. In essence, a seven-byte, 56-bit "Shifted Key Buffer" is transformed into an eight-byte, "Permutation Buffer" without altering the shifted Key. The arrows in Figure 9 indicate a few of the translation steps. Only six bits of each byte of the Permutation Buffer are used; the two high-order bits of each byte are cleared. This means only 48 of the 56 Shifted Key Buffer bits are used in any one iteration.

Different microprocessor architectures would best implement this type of permutation in different ways. Most approaches would share the steps of Figure 10a:

- Initialize the Permutation Buffer to default state (ones or zeroes):
- Isolate the state of a bit of a byte from the Key Buffer. Depending on the CPU, this might be accomplished by rotating a word of the Key Buffer through a carry flag or testing a bit in memory or an accumulator against a mask byte:
- Perform a conditional jump based on the carry or zero flag if the Permutation Buffer default state is correct:
- Otherwise reverse the corresponding bit in the permutation buffer with logical operations and mask bytes.

Each step above may require several instructions. The last three steps must be repeated for all 48 bits. Most microprocessors would spend 300 to 3,000 microseconds on each of the 16 iterations.

Notice, though, that this flow chart looks a lot like Figure 8. The Boolean Processor can permute bits by simply moving them from the source to the carry to the destination—a total of two instructions taking four bytes and three microseconds per bit. Assume the Shifted Key Buffer and Permutation Buffer both reside in bit-addressable RAM, with the bits of the former assigned symbolic names SKB__1, SKB__2, . . . SKB__56, and that the bytes of the latter are named PB__1, . . . PB__8. Then working from Figure 9, the software for the permutation algorithm would be that of Example 1a. The total routine length would be 192 bytes, requiring 144 microseconds.

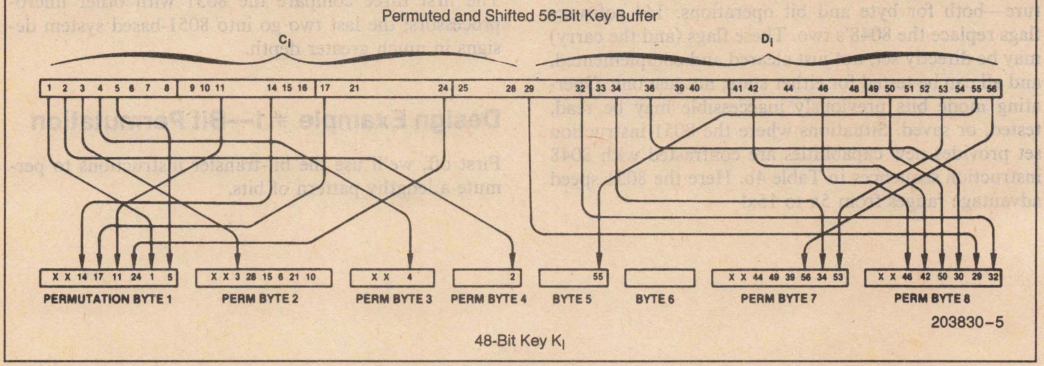


Figure 9. DES Key Schedule Transformation

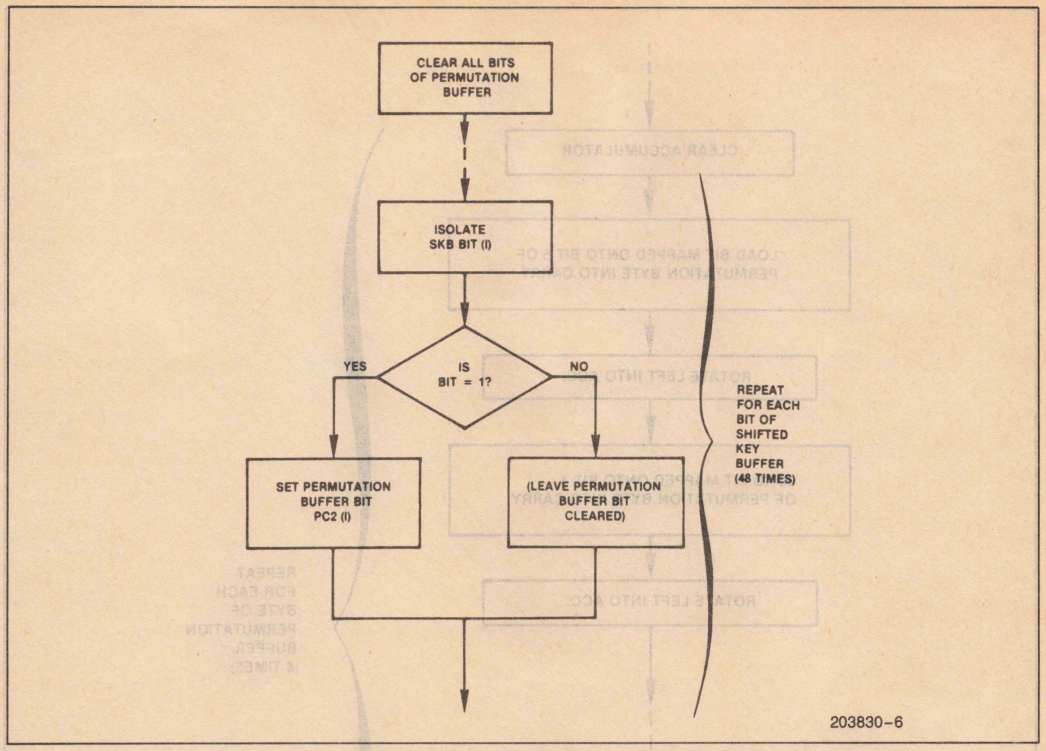


Figure 10a. Flowchart for Key Permutation Attempted with a Byte Processor

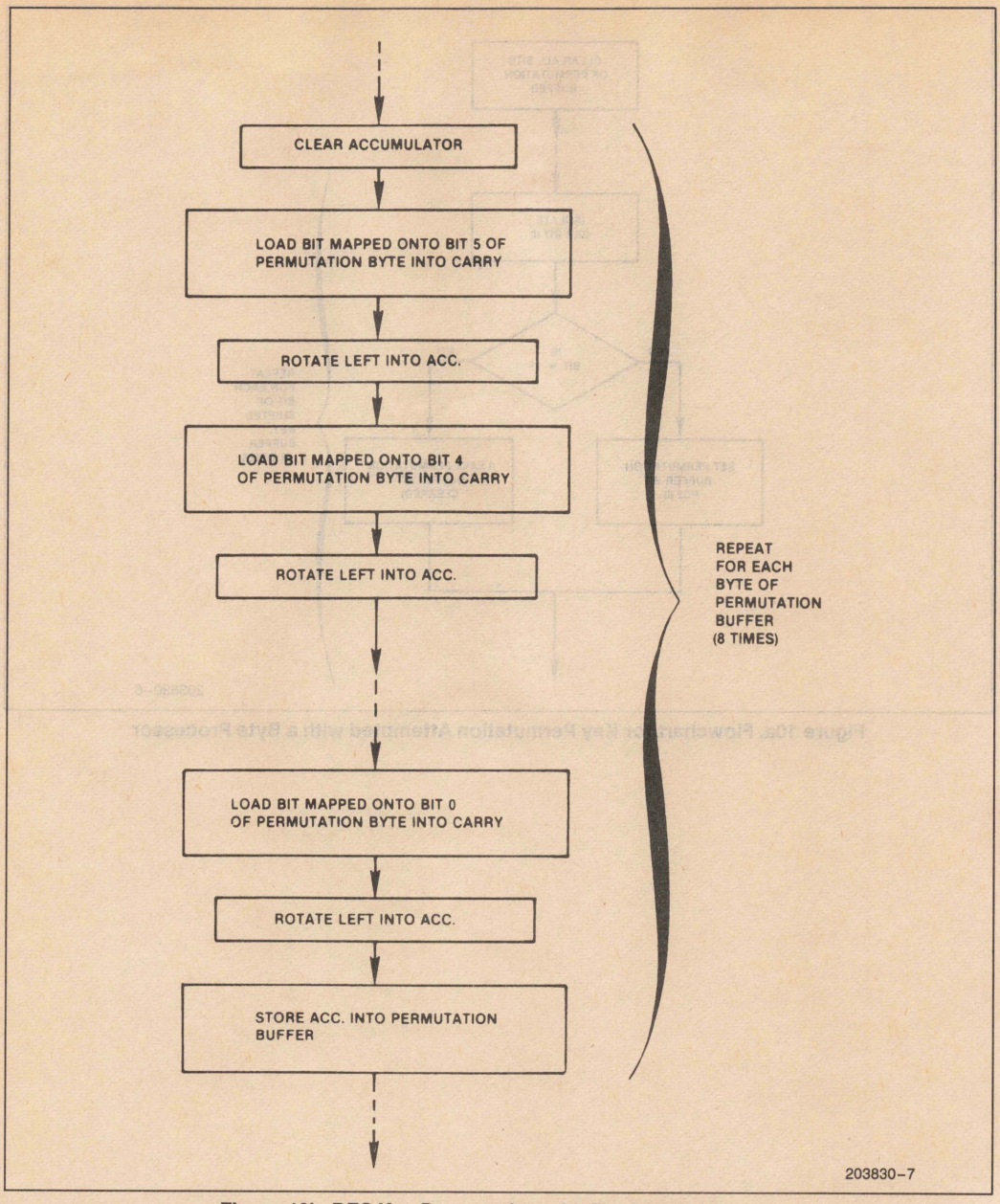


Figure 10b. DES Key Permutation with Boolean Processor

The algorithm of Figure 10b is just slightly more efficient in this time-critical application and illustrates the synergy of an integrated byte and bit processor. The bits needed for each byte of the Permutation Buffer are assimilated by loading each bit into the carry (1 μs.) and shifting it into the accumulator (1 μs.). Each byte is stored in RAM when completed. Forty-eight bits thus need a total of 112 instructions, some of which are listed in Example 1b.

Worst-case execution time would be 112 microseconds, since each instruction takes a single cycle. Routine length would also decrease, to 168 bytes. (Actually, in the context of the complete encryption algorithm, each permuted byte would be processed as soon as it is assimilated—saving memory and cutting execution time by another 8 μs.)

To date, most banking terminals and other systems using the DES have needed special boards or peripheral controller chips just for the encryption/decryption process, and still more hardware to form a serial bit stream for transmission (Figure 11a). An 8051 solution could pack most of the entire system onto the one chip (Figure 11b). The whole DES algorithm would require less than one-fourth of the on-chip program memory, with the remaining bytes free for operating the banking terminal (or whatever) itself.

Moreover, since transmission and reception of data is performed through the on-board UART, the unencrypted data (plaintext) never even exists outside the microcomputer! Naturally, this would afford a high degree of security from data interception.

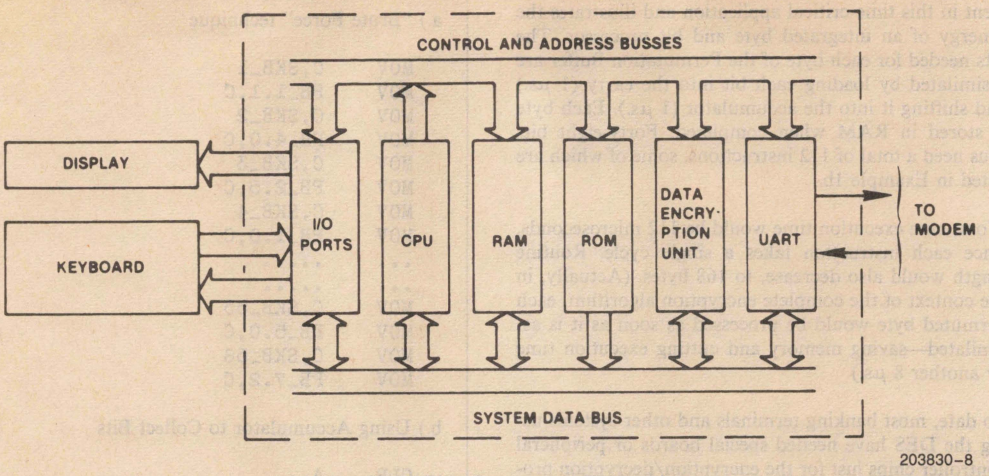
Example 1. DES Key Permutation Software.

a.) "Brute Force" technique

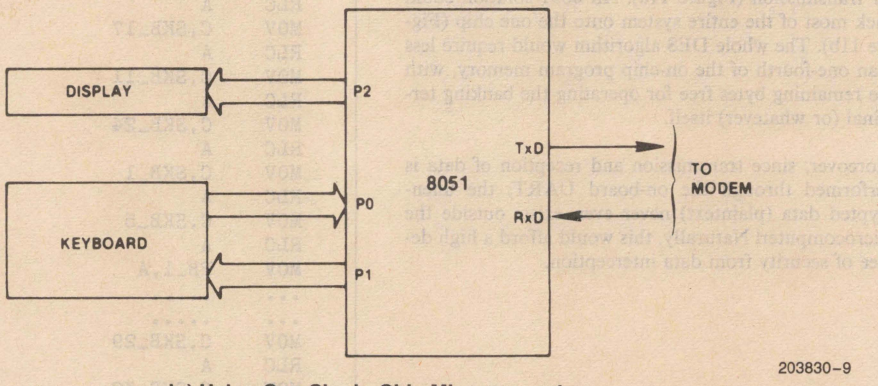
```
MOV     C,SKB_1
MOV     PB_1.1,C
MOV     C,SKB_2
MOV     PB_4.0,C
MOV     C,SKB_3
MOV     PB_2.5,C
MOV     C,SKB_4
MOV     PB_1.0,C
...     .....
...     .....
MOV     C,SKB_55
MOV     PB_5.0,C
MOV     C,SKB_56
MOV     PB_7.2,C
```

b.) Using Accumulator to Collect Bits

```
CLR     A
MOV     C,SKB_14
RLC     A
MOV     C,SKB_17
RLC     A
MOV     C,SKB_11
RLC     A
MOV     C,SKB_24
RLC     A
MOV     C,SKB_1
RLC     A
MOV     C,SKB_5
RLC     A
MOV     PB_1,A
...     .....
...     .....
MOV     C,SKB_29
RLC     A
MOV     C,SKB_32
RLC     A
MOV     PB_8,A
```



a.) Using Multi-Chip Processor Technology



b.) Using One Single-Chip Microcomputer

Figure 11. Secure Banking Terminal Block Diagram

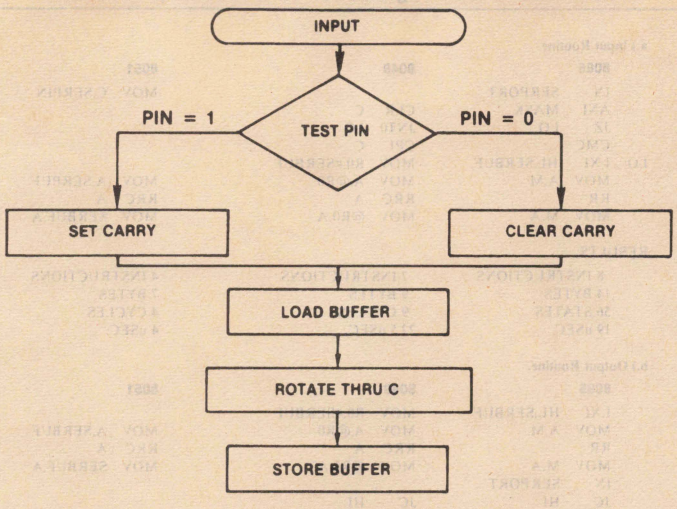
Design Example #2—Software Serial I/O

An exercise often imposed on beginning microcomputer students is to write a program simulating a UART. (See, for example, Application Notes AP24, AP29, and AP49.) Though doing this with the 8051 Family may appear to be a moot point (given that the hardware for a full UART is on-chip), it is still instructive to see how it would be done, and maintains a product line tradition.

As it turns out, the 8051 microcomputers can receive or transmit serial data via software very efficiently using the Boolean instruction set. Since any I/O pin may be a serial input or output, several serial links could be maintained at once.

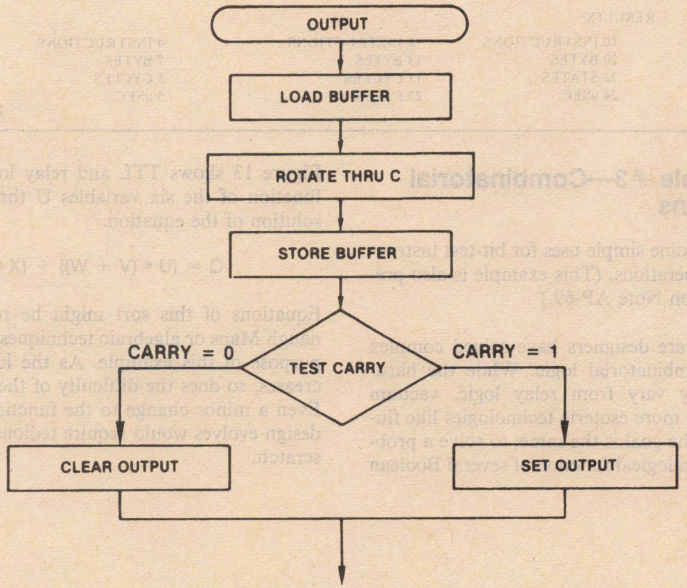
Figures 12a and 12b show algorithms for receiving or transmitting a byte of data. (Another section of program would invoke this algorithm eight times, synchronizing it with a start bit, clock signal, software delay, or timer interrupt.) Data is received by testing an input pin, setting the carry to the same state, shifting the carry into a data buffer, and saving the partial frame in internal RAM. Data is transmitted by shifting an output buffer through the carry, and generating each bit on an output pin.

A side-by-side comparison of the software for this common "bit-banging" application with three different microprocessor architectures is shown in Table 5a and 5b. The 8051 solution is more efficient than the others on every count!



203830-10

a.) Reception



203830-11

b.) Transmission

Figure 12. Serial I/O Algorithms

Table 5. Serial I/O Programs for Various Microprocessors

a.) Input Routine.

8085		8048		8051	
IN	SERPORT			MOV	C,SERPIN
ANI	MASK	CLR	C		
JZ	I.O	JNT0	I.O		
CMC		CPL	C		
I.O: LXI	HL,SERBUF	MOV	R0,#SERBUF		
MOV	A,M	MOV	A,@R0	MOV	A,SERBUF
RR		RRC	A	RRC	A
MOV	M,A	MOV	@R0,A	MOV	SERBUF,A
RESULTS:					
8 INSTRUCTIONS		7 INSTRUCTIONS		4 INSTRUCTIONS	
14 BYTES		9 BYTES		7 BYTES	
56 STATES		9 CYCLES		4 CYCLES	
19 uSEC.		22.5 uSEC.		4 uSEC.	

b.) Output Routine.

8085		8048		8051	
LXI	HL,SERBUF	MOV	R0,#SERBUF		
MOV	A,M	MOV	A,@R0	MOV	A,SERBUF
RR		RRC	A	RRC	A
MOV	M,A	MOV	@R0,A	MOV	SERBUF,A
IN	SERPORT				
JC	HI	JC	HI		
I.O: ANI	NOT MASK	ANL	SERPRT,#NOT MASK	MOV	SERPIN,C
JMP	CNT	JMP	CNT		
HI: ORI	MASK	HI: ORL	SERPRT,#MASK		
CNT: OUT	SERPORT	CNT:			
RESULTS:					
10 INSTRUCTIONS		8 INSTRUCTIONS		4 INSTRUCTIONS	
20 BYTES		13 BYTES		7 BYTES	
72 STATES		11 CYCLES		5 CYCLES	
24 uSEC.		27.5 uSEC.		5 uSEC.	

203830-30

Design Example #3—Combinatorial Logic Equations

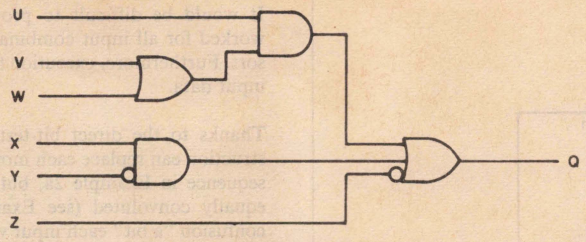
Next we'll look at some simple uses for bit-test instructions and logical operations. (This example is also presented in Application Note AP-69.)

Virtually all hardware designers have solved complex functions using combinatorial logic. While the hardware involved may vary from relay logic, vacuum tubes, or TTL or to more esoteric technologies like fluidics, in each case the goal is the same: to solve a problem represented by a logical function of several Boolean variables.

Figure 13 shows TTL and relay logic diagrams for a function of the six variables U through Z. Each is a solution of the equation.

$$Q = (U \bullet (V + W)) + (X \bullet \overline{Y}) + \overline{Z}$$

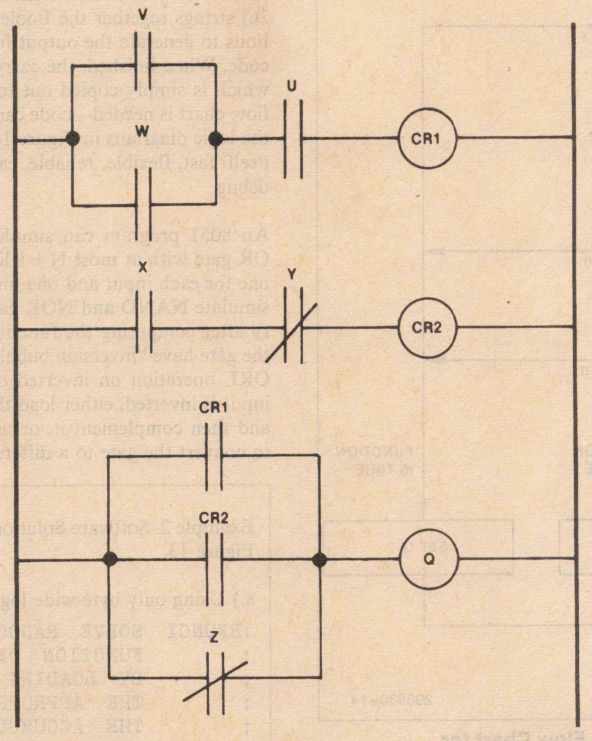
Equations of this sort might be reduced using Karnaugh Maps or algebraic techniques, but that is not the purpose of this example. As the logic complexity increases, so does the difficulty of the reduction process. Even a minor change to the function equations as the design evolves would require tedious re-reduction from scratch.



203830-12

$Q = (U \bullet (V + W)) + (X \bullet \overline{Y}) + \overline{Z}$

a.) Using TTL



203830-13

b.) Using Relay Logic

Figure 13. Hardware Implementations of Boolean Functions

For the sake of comparison we will implement this function three ways, restricting the software to three proper subsets of the MCS-51 instruction set. We will also assume that U and V are input pins from different input ports, W and X are status bits for two peripheral controllers, and Y and Z are software flags set up earlier in the program. The end result must be written to an output pin on some third port. The first two implementations follow the flow-chart shown in Figure 14. Program flow would embark on a route down a test-and-branch tree and leaves either the "True" or "Not True" exit ASAP—as soon as the proper result has been determined. These exits then rewrite the output port with the result bit respectively one or zero.

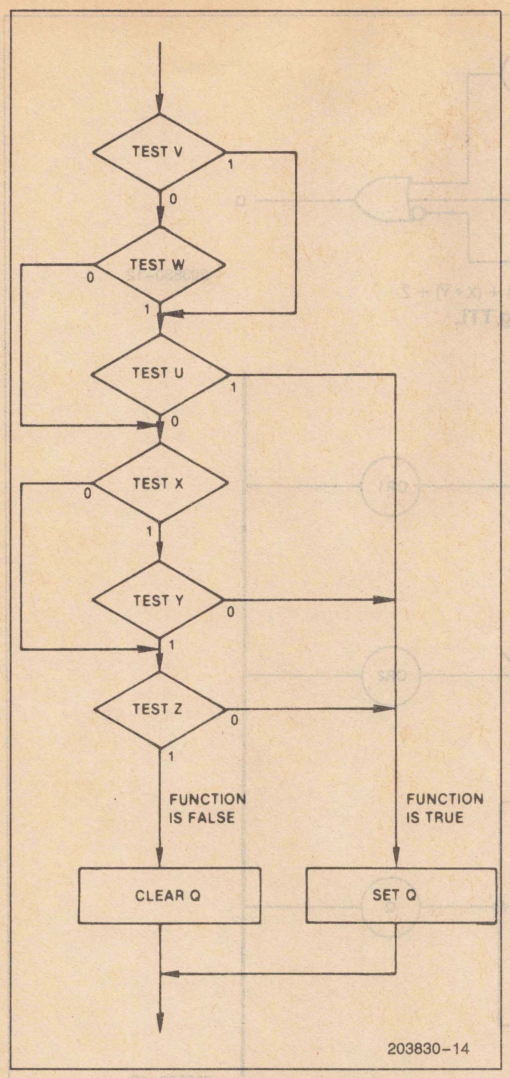


Figure 14. Flow Chart for Tree-Branching Algorithm

Other digital computers must solve equations of this type with standard word-wide logical instructions and conditional jumps. So for the first implementation, we won't use any generalized bit-addressing instructions. As we shall soon see, being constrained to such an instruction subset produces somewhat sloppy software solutions. MCS-51 mnemonics are used in Example 2a: other machines might further cloud the situation by requiring operation-specific mnemonics like INPUT, OUTPUT, LOAD, STORE, etc., instead of the MOV mnemonic used for all variable transfers in the 8051 instruction set.

The code which results is cumbersome and error prone. It would be difficult to prove whether the software worked for all input combinations in programs of this sort. Furthermore, execution time will vary widely with input data.

Thanks to the direct bit-test operations, a single instruction can replace each move mask conditional jump sequence in Example 2a, but the algorithm would be equally convoluted (see Example 2b). To lessen the confusion "a bit" each input variable is assigned a symbolic name.

A more elegant and efficient implementation (Example 2c) strings together the Boolean ANL and ORL functions to generate the output function with straight-line code. When finished, the carry flag contains the result, which is simply copied out to the destination pin. No flow chart is needed—code can be written directly from the logic diagrams in Figure 14. The result is simplicity itself: fast, flexible, reliable, easy to design, and easy to debug.

An 8051 program can simulate an N-input AND or OR gate with at most N + 1 lines of source program—one for each input and one line to store the results. To simulate NAND and NOR gates, complement the carry after computing the function. When some inputs to the gate have "inversion bubbles", perform the ANL or ORL operation on inverted operands. When the first input is inverted, either load the operand into the carry and then complement it, or use DeMorgan's Theorem to convert the gate to a different form.

Example 2. Software Solutions to Logic Function of Figure 13.

a.) Using only byte-wide logical instructions

```
:BFUNCI  SOLVE RANDOM LOGIC
;        FUNCTION OF 6 VARIABLES
;        BY LOADING AND MASKING
;        THE APPROPRIATE BITS IN
;        THE ACCUMULATOR. THEN
;        EXECUTING CONDITIONAL
;        JUMPS BASED ON ZERO
;        CONDITION. (APPROACH USED
;        BY BYTE-ORIENTED
;        ARCHITECTURES.) BYTE AND
;        MASK VALUES CORRESPOND TO
;        RESPECTIVE BYTE ADDRESS
;        AND BIT POSITIONS.
;
OUTBUF   DATA 22H
;OUTPUT PIN STATE MAP
;
```

```
TESTV:  MOV   A,P2
        ANL   A,#00000100B
        JNZ   TESTU
        MOV   A,TCON
        ANL   A,#00100000B
        JZ    TESTX
TESTU:  MOV   A,P1
        ANL   A,#00000010B
        JNZ   SETQ
TESTX:  MOV   A,TCON
        ANL   A,#00001000B
        JZ    TESTZ
        MOV   A,20H
        ANL   A,#00000001B
        JZ    SETQ
TESTZ:  MOV   A,21H
        ANL   A,#00000010B
        JZ    SETQ
CLRQ:   MOV   A,OUTBUF
        ANL   A,#11110111B
        JMP   OUTQ
SETQ:   MOV   A,OUTBUF
        ORL   A,#00001000B
OUTQ:   MOV   OUTBUF,A
        MOV   P3,A
```

b.) Using only bit-test instructions

```
:BFUNC2  SOLVE A RANDOM LOGIC
;        FUNCTION OF 6 VARIABLES
;        BY DIRECTLY POLLING EACH
;        BIT.  (APPROACH USING
;        MCS-51 UNIQUE BIT-TEST
;        INSTRUCTION CAPABILITY.)
;        SYMBOLS USED IN LOGIC
;        DIAGRAM ASSIGNED TO
;        CORRESPONDING 8x51 BIT
;        ADDRESSES.
;
U        BIT   P1.1
V        BIT   P2.2
W        BIT   TF0
X        BIT   IE1
Y        BIT   20H.0
Z        BIT   21H.1
Q        BIT   P3.3
;        ...   ....
TEST_V:  JB    V,TEST_U
         JNB   W,TEST_X
TEST_U:  JB    U,SET_Q
TEST_X:  JNB   X,TEST_Z
         JNB   Y,SET_Q
TEST_Z:  JNB   Z,SET_Q
CLR_Q:   CLR   Q
         JMP   NXTTST
SET_Q:   SETB  Q
NXTTST:(CONTINUATION OF
       :PROGRAM)
```

c.) Using logical operations on Boolean variables

```
:FUNC3   SOLVE A RANDOM LOGIC
;        FUNCTION OF 6 VARIABLES
;        USING STRAIGHT_LINE
;        LOGICAL INSTRUCTIONS ON
;        MCS-51 BOOLEAN VARIABLES.
;
MOV C,V
ORL C,W   ;OUTPUT OF OR GATE
ANL C,U   ;OUTPUT OF TOP AND GATE
MOV F0,C  ;SAVE INTERMEDIATE STATE
MOV C,X
ANL C,Y   ;OUTPUT OF BOTTOM AND GATE
ORL C,F0  ;INCLUDE VALUE SAVED ABOVE
ORL C,Z   ;INCLUDE LAST INPUT
          ;VARIABLE
MOV Q,C   ;OUTPUT COMPUTED RESULT
```

An upper-limit can be placed on the complexity of software to simulate a large number of gates by summing the total number of inputs and outputs. The *actual* total should be somewhat shorter, since calculations can be "chained," as shown. The output of one gate is often the first input to another, bypassing the intermediate variable to eliminate two lines of source.

Design Example #4—Automotive Dashboard Functions

Now let's apply these techniques to designing the software for a complete controller system. This application is patterned after a familiar real-world application which isn't nearly as trivial as it might first appear: automobile turn signals.

Imagine the three position turn lever on the steering column as a single-pole, triple-throw toggle switch. In its central position all contacts are open. In the up or down positions contacts close causing corresponding lights in the rear of the car to blink. So far very simple.

Two more turn signals blink in the front of the car, and two others in the dashboard. All six bulbs flash when an emergency switch is closed. A thermo-mechanical relay (accessible under the dashboard in case it wears out) causes the blinking.

Applying the brake pedal turns the tail light filaments on constantly . . . unless a turn is in progress, in which case the blinking tail light is not affected. (Of course, the front turn signals and dashboard indicators are not affected by the brake pedal.) Table 6 summarizes these operating modes.

Table 6. Truth Table for Turn-Signal Operation

Input Signals				Output Signals			
Brake Switch	Emerg. Switch	Left Turn Switch	Right Turn Switch	Left Front & Dash	Right Front & Dash	Left Rear	Right Rear
0	0	0	0	Off	Off	Off	Off
0	0	0	1	Off	Blink	Off	Blink
0	0	1	0	Blink	Off	Blink	Off
0	1	0	0	Blink	Blink	Blink	Blink
0	1	0	1	Blink	Blink	Blink	Blink
0	1	1	0	Blink	Blink	Blink	Blink
1	0	0	0	Off	Off	On	On
1	0	0	1	Off	Blink	On	Blink
1	0	1	0	Blink	Off	Blink	On
1	1	0	0	Blink	Blink	On	On
1	1	0	1	Blink	Blink	On	Blink
1	1	1	0	Blink	Blink	Blink	On

But we're not done yet. Each of the exterior turn signal (but not the dashboard) bulbs has a second, somewhat dimmer filament for the parking lights. Figure 15 shows TTL circuitry which could control all six bulbs. The signals labeled "High Freq." and "Low Freq." represent two square-wave inputs. Basically, when one of the turn switches is closed or the emergency switch is activated the low frequency signal (about 1 Hz) is gated through to the appropriate dashboard indicator(s) and turn signal(s). The rear signals are also activated when the brake pedal is depressed provided a turn is not being made in the same direction. When the parking light switch is closed the higher frequency oscillator is gated to each front and rear turn signal, sustaining a low-intensity background level. (This is to eliminate the need for additional parking light filaments.)

In most cars, the switching logic to generate these functions requires a number of multiple-throw contacts. As many as 18 conductors thread the steering column of some automobiles solely for turn-signal and emergency blinker functions. (The author discovered this recently to his astonishment and dismay when replacing the whole assembly because of one burned contact.)

A multiple-conductor wiring harness runs to each corner of the car, behind the dash, up the steering column, and down to the blinker relay below. Connectors at each termination for each filament lead to extra cost and labor during construction, lower reliability and safety, and more costly repairs. And considering the system's present complexity, increasing its reliability or detecting failures would be quite difficult.

There are two reasons for going into such painful detail describing this example. First, to show that the messiest part of many system designs is determining what the controller should do. Writing the software to solve these functions will be comparatively easy. Secondly, to show the many potential failure points in the system. Later we'll see how the peripheral functions and intelligence built into a microcomputer (with a little creativity) can greatly reduce external interconnections and mechanical part count.

The Single-Chip Solution

The circuit shown in Figure 16 indicates five input pins to the five input variables—left-turn select, right-turn select, brake pedal down, emergency switch on, and parking lights on. Six output pins turn on the front, rear, and dashboard indicators for each side. The microcomputer implements all logical functions through software, which periodically updates the output signals as time elapses and input conditions change.

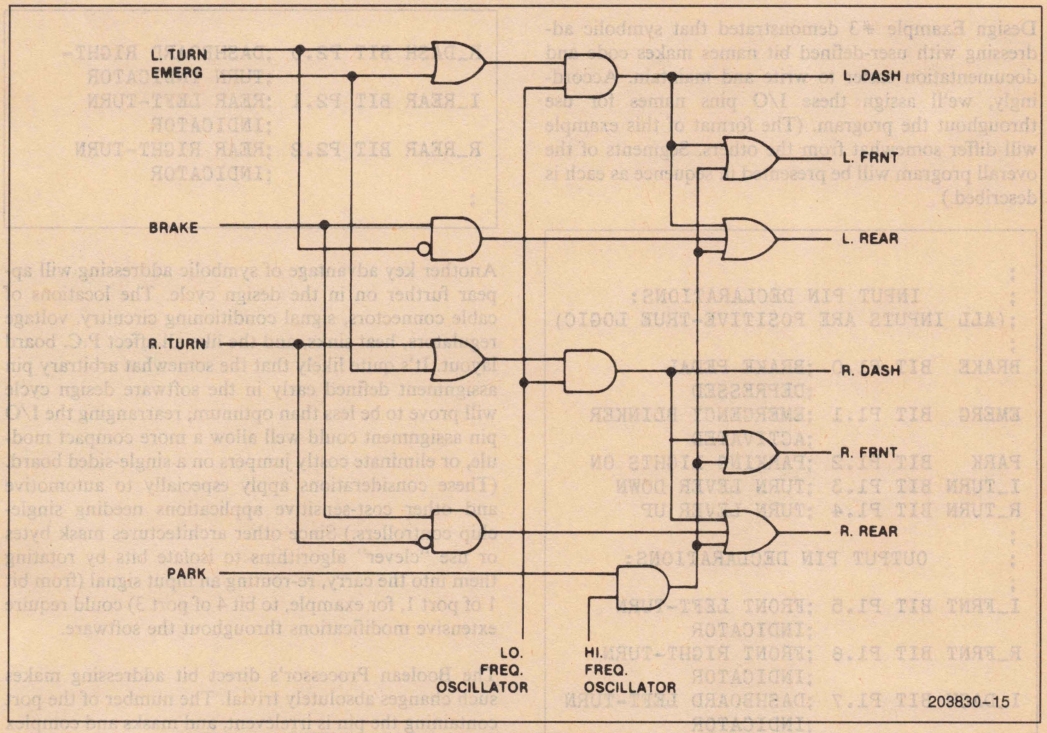


Figure 15. TTL Logic Implementation of Automotive Turn Signals

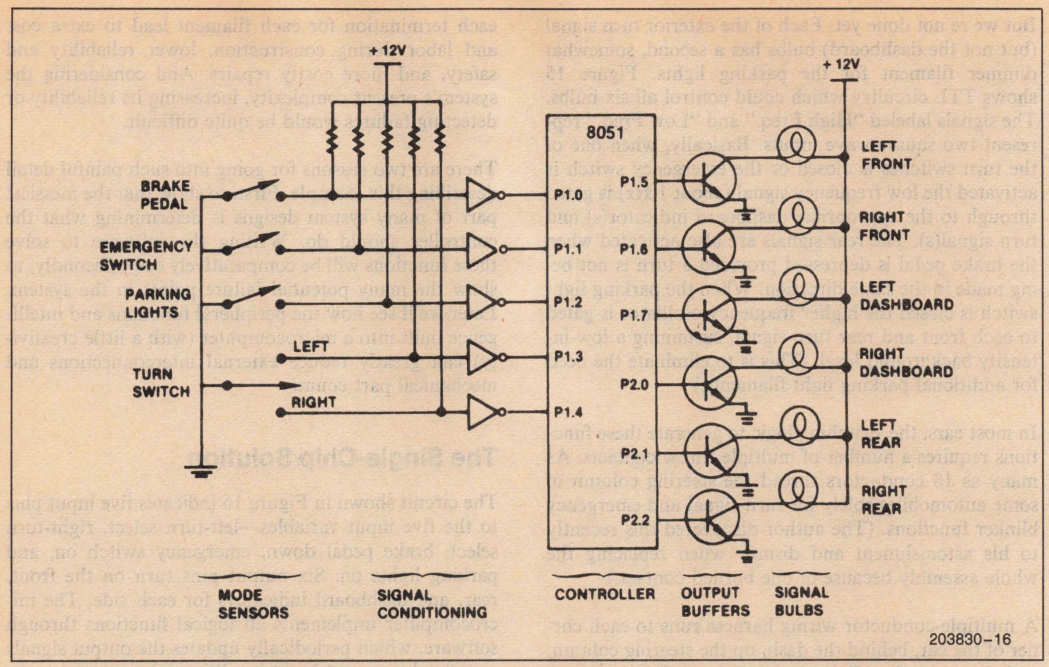


Figure 16. Microcomputer Turn-Signal Connections

Design Example #3 demonstrated that symbolic addressing with user-defined bit names makes code and documentation easier to write and maintain. Accordingly, we'll assign these I/O pins names for use throughout the program. (The format of this example will differ somewhat from the others. Segments of the overall program will be presented in sequence as each is described.)

```
;
;       INPUT PIN DECLARATIONS:
;(ALL INPUTS ARE POSITIVE-TRUE LOGIC)
;
BRAKE   BIT P1.0 ;BRAKE PEDAL
                 ;DEPRESSED
EMERG   BIT P1.1 ;EMERGENCY BLINKER
                 ;ACTIVATED
PARK    BIT P1.2 ;PARKING LIGHTS ON
I_TURN  BIT P1.3 ;TURN LEVER DOWN
R_TURN  BIT P1.4 ;TURN LEVER UP
;
;       OUTPUT PIN DECLARATIONS:
;
I_FRNT  BIT P1.5 ;FRONT LEFT-TURN
                 ;INDICATOR
R_FRNT  BIT P1.6 ;FRONT RIGHT-TURN
                 ;INDICATOR
I_DASH  BIT P1.7 ;DASHBOARD LEFT-TURN
                 ;INDICATOR
```

```
R_DASH  BIT P2.0 ;DASHBOARD RIGHT-
                 ;TURN INDICATOR
I_REAR  BIT P2.1 ;REAR LEFT-TURN
                 ;INDICATOR
R_REAR  BIT P2.2 ;REAR RIGHT-TURN
                 ;INDICATOR
;
```

Another key advantage of symbolic addressing will appear further on in the design cycle. The locations of cable connectors, signal conditioning circuitry, voltage regulators, heat sinks, and the like all affect P.C. board layout. It's quite likely that the somewhat arbitrary pin assignment defined early in the software design cycle will prove to be less than optimum; rearranging the I/O pin assignment could well allow a more compact module, or eliminate costly jumpers on a single-sided board. (These considerations apply especially to automotive and other cost-sensitive applications needing single-chip controllers.) Since other architectures mask bytes or use "clever" algorithms to isolate bits by rotating them into the carry, re-routing an input signal (from bit 1 of port 1, for example, to bit 4 of port 3) could require extensive modifications throughout the software.

The Boolean Processor's direct bit addressing makes such changes absolutely trivial. The number of the port containing the pin is irrelevent, and masks and complex

program structures are not needed. Only the initial Boolean variable declarations need to be changed; ASM51 automatically adjusts all addresses and symbolic references to the reassigned variables. The user is assured that no additional debugging or software verification will be required.

```
;           ...          .....
;INTERRUPT RATE SUBDIVIDER
SUB_DIV    DATA         20H
;HIGH-FREQUENCY OSCILLATOR BIT
HI_FREQ    BIT          SUB_DIV,0
;LOW-FREQUENCY OSCILLATOR BIT
LO_FREQ    BIT          SUB_DIV,7
;           ...
           ORG          0000H
JMP        INIT
;           ...          .....
           ORG          100H
;PUT TIMER 0 IN MODE 1
INIT;      MOV          TMOD,#00000001B
;INITIALIZE TIMER REGISTERS
           MOV          TL0,#0
           MOV          TH0,#-16
;SUBDIVIDE INTERRUPT RATE BY 244
           MOV          SUB_DIV,#244
;ENABLE TIMER INTERRUPTS
           SETB         ET0
;GLOBALLY ENABLE ALL INTERRUPTS
           SETB         EA
;START TIMER
           SETB         TR0
;
;(CONTINUE WITH BACKGROUND PROGRAM)
;
;PUT TIMER 0 IN MODE 1
;INITIALIZE TIMER REGISTERS

;SUBDIVIDE INTERRUPT RATE BY 244
;ENABLE TIMER INTERRUPTS
;GLOBALLY ENABLE ALL INTERRUPTS
;START TIMER
```

Timer 0 (one of the two on-chip timer counters) replaces the thermo-mechanical blinker relay in the dashboard controller. During system initialization it is configured as a timer in mode 1 by setting the least significant bit of the timer mode register (TMOD). In this configuration the low-order byte (TL0) is incremented every machine cycle, overflowing and incrementing the high-order byte (TH0) every 256 μs. Timer interrupt 0 is enabled so that a hardware interrupt will occur each time TH0 overflows.

An eight-bit variable in the bit-addressable RAM array will be needed to further subdivide the interrupts via software. The lowest-order bit of this counter toggles very fast to modulate the parking lights: bit 7 will be "tuned" to approximately 1 Hz for the turn- and emergency-indicator blinking rate.

Loading TH0 with -16 will cause an interrupt after 4.096 ms. The interrupt service routine reloads the high-order byte of timer 0 for the next interval, saves the CPU registers likely to be affected on the stack, and then decrements SUB_DIV. Loading SUB_DIV. with 244 initially and each time it decrements to zero will produce a 0.999 second period for the highest-order bit.

```
ORG  000BH  ;TIMER 0 SERVICE VECTOR
MOV  TH0,#-16
PUSH PSW
PUSH ACC
PUSH B
DJNZ SUB_DIV,TOSERV
MOV  SUB_DIV,#244
```

The code to sample inputs, perform calculations, and update outputs—the real "meat" of the signal controller algorithm—may be performed either as part of the interrupt service routine or as part of a background program loop. The only concern is that it must be executed at least serveral dozen times per second to prevent parking light flickering. We will assume the former case, and insert the code into the timer 0 service routine.

First, notice from the logic diagram (Figure 15) that the subterm (PARK • H_FREQ), asserted when the parking lights are to be on dimly, figures into four of the six output functions. Accordingly, we will first compute that term and save it in a temporary location named "DIM". The PSW contains two general purpose flags: F0, which corresponds to the 8048 flag of the same name, and PSW.1. Since the PSW has been saved and will be restored to its previous state after servicing the interrupt, we can use either bit for temporary storage.

```
  DIM BIT   PSW.1 ;DECLARE TEMP
                  ;STORAGE FLAG
; ... .....
  MOV C,PARK      ;GATE PARKING
                  ;LIGHT SWITCH
  ANL HI_FREQ     ;WITH HIGH
                  ;FREQUENCY
                  ;SIGNAL
  MOV DIM,C       ;AND SAVE IN
                  ;TEMP. VARIABLE
```

This simple three-line section of code illustrates a remarkable point. The software indicates in very abstract terms exactly what function is being performed, inde-

pendent of the hardware configuration. The fact that these three bits include an input pin, a bit within a program variable, and a software flag in the PSW is totally invisible to the programmer.

Now generate and output the dashboard left turn signal.

```
;
  MOV C,L_TURN      ;SET CARRY IF
                    ;TURN
  ORL C,EMERG       ;OR EMERGENCY
                    ;SELECTED
  ANL C,LO_FREQ     ;GATE IN 1 HZ
                    ;SIGNAL
  MOV I_DASH,C      ;AND OUTPUT TO
                    ;DASHBOARD
```

To generate the left front turn signal we only need to add the parking light function in F0. But notice that the function in the carry will also be needed for the rear signal. We can save effort later by saving its current state in F0.

```
;
  MOV F0,C          ;SAVE FUNCTION
                    ;SO FAR
  ORL C,DIM         ;ADD IN PARKING
                    ;LIGHT FUNCTION
  MOV L_FRNT,C      ;AND OUTPUT TO
                    ;TURN SIGNAL
```

Finally, the rear left turn signal should also be on when the brake pedal is depressed, provided a left turn is not in progress.

```
MOV C,BRAKE       ;GATE BRAKE
                  ;PEDAL SWITCH
ANL C,L_TURN      ;WITH TURN
                  ;LEVER
ORL C,F0          ;INCLUDE TEMP.
                  ;VARIABLE FROM DASH
ORL C,DIM         ;AND PARKING
                  ;LIGHT FUNCTION
MOV L_REAR,C      ;AND OUTPUT TO
                  ;TURN SIGNAL
```

Now we have to go through a similar sequence for the right-hand equivalents to all the left-turn lights. This also gives us a chance to see how the code segments above look when combined.

```
MOV C.R_TURN      ;SET CARRY H-
                  ;TURN
ORL C.EMERG       ;OR EMERGENCY
                  ;SELECTED
ANL C,LO_FREQ     ;IF SO. GATE IN 1
                  ;HZ SIGNAL
MOV R_DASH.C      ;AND OUTPUT TO
                  ;DASHBOARD
MOV F0.C          ;SAVE FUNCTION
                  ;SO FAR
ORL C.DIM         ;ADD IN PARKING
                  ;LIGHT FUNCTION
MOV R_FRNT.C      ;AND OUTPUT TO
                  ;TURN SIGNAL
MOV C.BRAKE       ;GATE BRAKE
                  ;PEDAL SWITCH
ANL C. R_TURN     ;WITH TURN
                  ;LEVER
ORL C.F0          ;INCLUDE TEMP.
                  ;VARIABLE FROM
                  ;DASH
ORL C.DIM         ;AND PARKING
                  ;LIGHT FUNCTION
MOV R_REAR.C      ;AND OUTPUT TO
                  ;TURN SIGNAL
```

(The perceptive reader may notice that simply rearranging the steps could eliminate one instruction from each sequence.)

Now that all six bulbs are in the proper states, we can return from the interrupt routine, and the program is finished. This code essentially needs to reverse the status saving steps at the beginning of the interrupt.

Table 7. Non-Trivial Duty Cycles

Sub_Div Bits								Duty Cycles						
7	6	5	4	3	2	1	0	12.5%	25.0%	37.5%	50.0%	62.5%	75.0%	87.5%
X	X	X	X	X	0	0	0	Off	Off	Off	Off	Off	Off	Off
X	X	X	X	X	0	0	1	Off	Off	Off	Off	Off	Off	On
X	X	X	X	X	0	1	0	Off	Off	Off	Off	Off	On	On
X	X	X	X	X	0	1	1	Off	Off	Off	Off	On	On	On
X	X	X	X	X	1	0	0	Off	Off	Off	On	On	On	On
X	X	X	X	X	1	0	1	Off	Off	On	On	On	On	On
X	X	X	X	X	1	1	0	Off	On	On	On	On	On	On
X	X	X	X	X	1	1	1	On	On	On	On	On	On	On

```
POP  B              ;RESTORE CPU
                    ;REGISTERS.
POP ACC
POP PSW
RETI
```

Program Refinements. The luminescence of an incandescent light bulb filament is generally non-linear: the 50% duty cycle of HI_FREQ may not produce the desired intensity. If the application requires, duty cycles of 25%, 75%, etc. are easily achieved by ANDing and ORing in additional low-order bits of SUB_DIV. For example, 30 H/ signals of seven different duty cycles could be produced by considering bits 2–0 as shown in Table 7. The only software change required would be to the code which sets-up variable DIM;

```
MOV C,SUB_DIV.1 ;START WITH 50
                ;PERCENT
ANL C,SUB_DIV.0 ;MASK DOWN TO 25
                ;PERCENT
ORL C,SUB_DIV.2 ;AND BUILD BACK TO
                ;62 PERCENT
MOV DIM,C       ;DUTY CYCLE FOR
                ;PARKING LIGHTS.
```

Interconnections increase cost and decrease reliability. The simple buffered pin-per-function circuit in Figure 16 is insufficient when many outputs require higher-than-TTL drive levels. A lower-cost solution uses the 8051 serial port in the shift-register mode to augment I/O. In mode 0, writing a byte to the serial port data buffer (SBUF) causes the data to be output sequentially through the "RXD" pin while a burst of eight clock pulses is generated on the "TXD" pin. A shift register connected to these pins (Figure 17) will load the data byte as it is shifted out. A number of special peripheral driver circuits combining shift-register inputs with high drive level outputs have been introduced recently.

Cascading multiple shift registers end-to-end will expand the number of outputs even further. The data rate in the I/O expansion mode is one megabaud, or 8 μs. per byte. This is the mode which the serial port defaults to following a reset, so no initialization is required.

The software for this technique uses the B register as a "map" corresponding to the different output functions. The program manipulates these bits instead of the output pins. After all functions have been calculated the B register is shifted by the serial port to the shift-register driver. (While some outputs may glitch as data is shifted through them, at 1 Megabaud most people wouldn't notice. Some shift registers provide an "enable" bit to hold the output states while new data is being shifted in.)

This is where the earlier decision to address bits symbolically throughout the program is going to pay off. This major I/O restructuring is nearly as simple to implement as rearranging the input pins. Again, only the bit declarations need to be changed.

```
I_FRNT BIT B.0  ;FRONT LEFT-TURN
                ;INDICATOR
R_FRNT BIT B.1  ;FRONT RIGHT-TURN
                ;INDICATOR
I_DASH BIT B.2  ;DASHBOARD LEFT-TURN
                ;INDICATOR
R_DASH BIT B.3  ;DASHBOARD RIGHT-TURN
                ;INDICATOR
I_REAR BIT B.4  ;REAR LEFT-TURN
                ;INDICATOR
R_REAR BIT B.5  ;REAR RIGHT-TURN
                ;INDICATOR
```

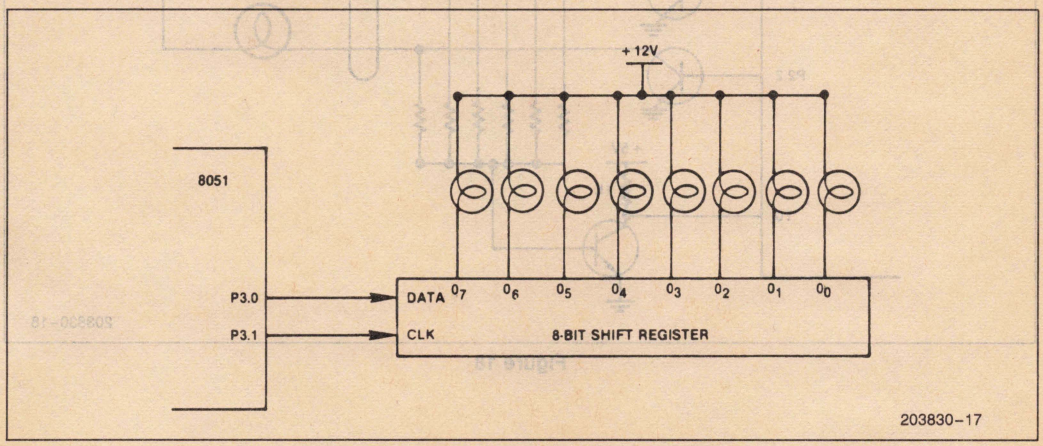


Figure 17. Output Expansion Using Serial Port

The original program to compute the functions need not change. After computing the output variables, the control map is transmitted to the buffered shift register through the serial port.

```
MOV SBUF,B ;LOAD BUFFER AND TRANSMIT
```

The Boolean Processor solution holds a number of advantages over older methods. Fewer switches are required. Each is simpler, requiring fewer poles and lower current contacts. The flasher relay is eliminated entirely. Only six filaments are driven, rather than 10. The wiring harness is therefore simpler and less expensive—one conductor for each of the six lamps and each of the five sensor switches. The fewer conductors use far fewer connectors. The whole system is more reliable.

And since the system is much simpler it would be feasible to implement redundancy and or fault detection on the four main turn indicators. Each could still be a standard double filament bulb, but with the filaments driven in parallel to tolerate single-element failures.

Even with redundancy, the lights will eventually fail. To handle this inescapable fact current or voltage sensing circuits on each main drive wire can verify that each bulb and its high-current driver is functioning properly. Figure 18 shows one such circuit.

Assume all of the lights are turned on except one: i.e., all but one of the collectors are grounded. For the bulb which is turned off, if there is continuity from +12V through the bulb base and filament, the control wire, all connectors, and the P.C. board traces, and if the transistor is indeed not shorted to ground, then the collector will be pulled to +12V. This turns on the base of Q8 through the corresponding resistor, and grounds the input pin, verifying that the bulb circuit is operational. The continuity of each circuit can be checked by software in this way.

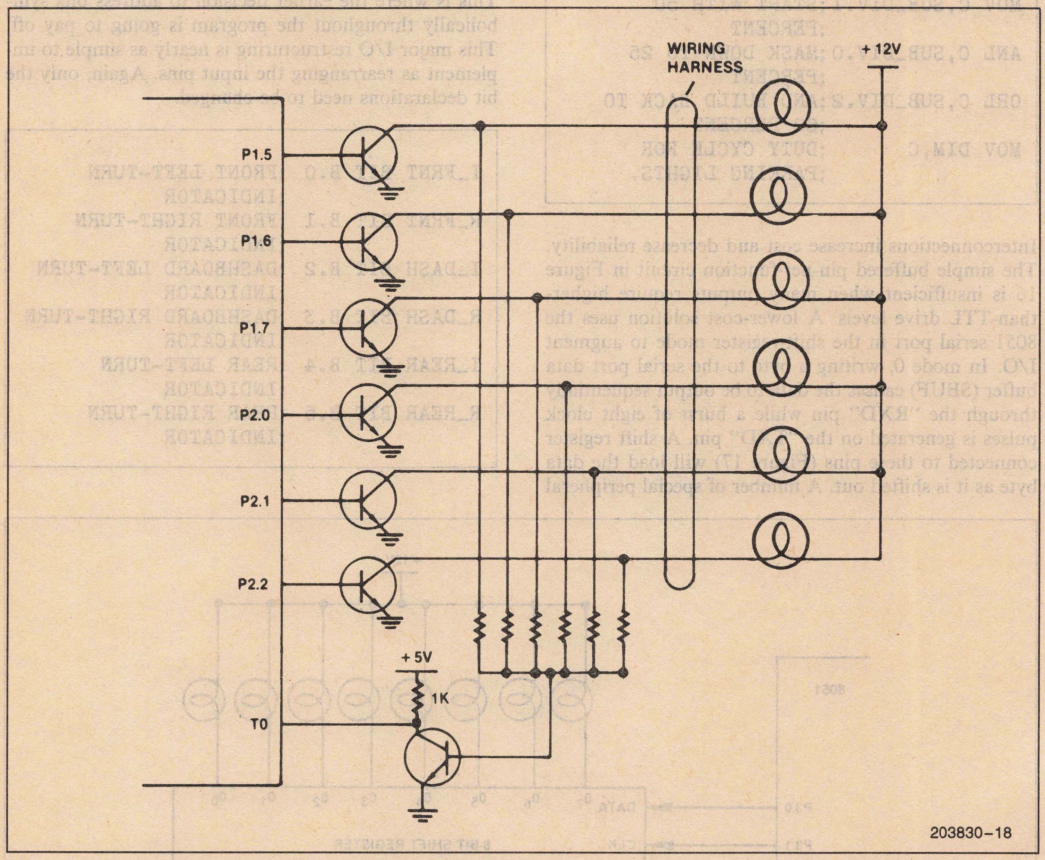


Figure 18

Now turn *all* the bulbs on, grounding all the collectors. Q7 should be turned off, and the Test pin should be high. However, a control wire shorted to +12V or an open-circuited drive transistor would leave one of the collectors at the higher voltage even now. This too would turn on Q7, indicating a different type of failure. Software could perform these checks once per second by executing the routine every time the software counter SUB_DIV is reloaded by the interrupt routine.

```
DJNZ SUB_DIV,TOSERV
MOV SUB_DIV,#244      ;RELOAD COUNTER
ORL P1,#11100000B     ;SET CONTROL
                      ;OUTPUTS HIGH
ORL P2,#00000111B
CLR I_FRNT            ;FLOAT DRIVE
                      ;COLLECTOR
JB  T0,FAULT          ;T0 SHOULD BE
                      ;PULLED LOW
SETB L_FRNT           ;PULL COLLECTOR
                      ;BACK DOWN
    CLR  L_DASH
    JB   T0,FAULT
    SETB L_DASH
    CLR  L_REAR
    JB   T0,FAULT
    SETB L_REAR
    CLR  R_FRNT
    JB   T0,FAULT
    SETB R_FRNT
    CLR  R_DASH
    JB   T0,FAULT
    SETB R_DASH
    CLR  R_REAR
    JB   T0,FAULT
    SETB R_REAR
;
;WITH ALL COLLECTORS GROUNDED. T0
 SHOULD BE HIGH
;IF SO. CONTINUE WITH INTERRUPT
 ROUTINE.
  JB   T0,TOSERV
FAULT:                ;ELECTRICAL
                      ;FAILURE
                      ;PROCESSING
                      ;ROUTINE
                      ;(LEFT TO
                      ;READER'S
                      ;IMAGINATION)
TOSERV:               ;CONTINUE WITH
                      ;INTERRUPT
                      ;PROCESSING
;
;
```

The complete assembled program listing is printed in Appendix A. The resulting code consists of 67 program statements, not counting declarations and comments, which assemble into 150 bytes of object code. Each pass through the service routine requires (coincidently) 67 μs plus 32 μs once per second for the electrical test. If executed every 4 ms as suggested this software would typically reduce the throughput of the background program by less than 2%.

Once a microcomputer has been designed into a system, new features suddenly become virtually free. Software could make the emergency blinkers flash alternately or at a rate faster than the turn signals. Turn signals could override the emergency blinkers. Adding more bulbs would allow multiple tail light sequencing and syncopation—true flash factor, so to speak.

Design Example #5—Complex Control Functions

Finally, we'll mix byte and bit operations to extend the use of 8051 into extremely complex applications.

Programmers can arbitrarily assign I/O pins to input and output functions only if the total does not exceed 32, which is insufficient for applications with a very large number of input variables. One way to expand the number of inputs is with a technique similar to multiplexed-keyboard scanning.

Figure 19 shows a block diagram for a moderately complex programmable industrial controller with the following characteristics:

- 64 input variable sensors:
- 12 output signals:
- Combinational and sequential logic computations:
- Remote operation with communications to a host processor via a high-speed full-duplex serial link:
- Two prioritized external interrupts:
- Internal real-time and time-of-day clocks.

While many microprocessors could be programmed to provide these capabilities with assorted peripheral support chips, an 8051 microcomputer needs no other integrated circuits!

The 64 input sensors are logically arranged as an 8x8 matrix. The pins of Port 1 sequentially enable each column of the sensor matrix: as each is enabled Port 0 reads in the state of each sensor in that column. An eight-byte block in bit-addressable RAM remembers the data as it is read in so that after each complete scan cycle there is an internal map of the current state of all sensors. Logic functions can then directly address the elements of the bit map.

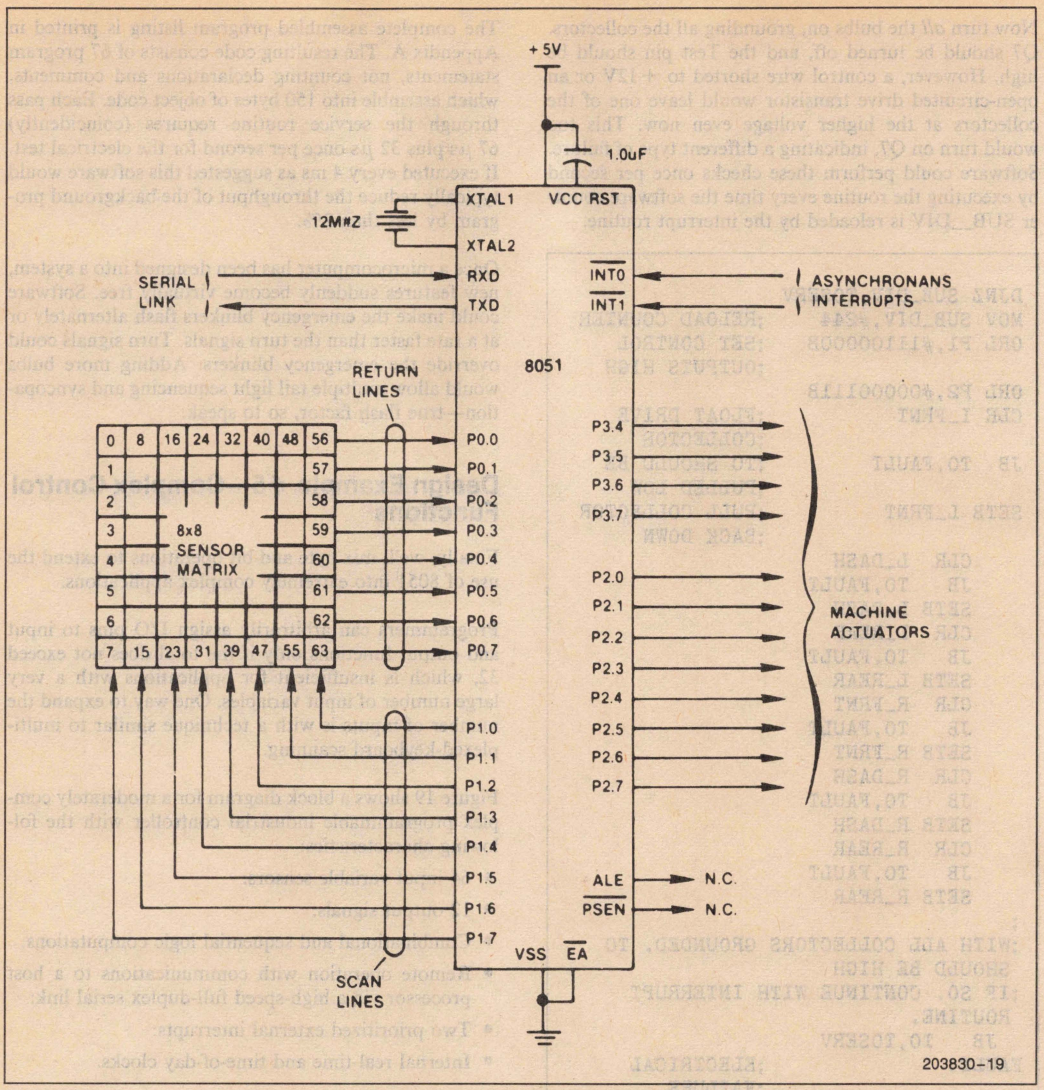


Figure 19. Block Diagram of 64-Input Machine Controller

The computer's serial port is configured as a nine-bit UART, transferring data at 17,000 bytes-per-second. The ninth bit may distinguish between address and data bytes.

The 8051 serial port can be configured to detect bytes with the address bit set, automatically ignoring all others. Pins INT0 and INT1 are interrupts configured respectively as high-priority, falling-edge triggered and low-priority, low-level triggered. The remaining 12 I/O pins output TTL-level control signals to 12 actuators.

There are several ways to implement the sensor matrix circuitry, all logically similar. Figure 20a shows one possibility. Each of the 64 sensors consists of a pair of simple switch contacts in series with a diode to permit multiple contact closures throughout the matrix.

The scan lines from Port 1 provide eight un-encoded active-high scan signals for enabling columns of the matrix. The return lines on rows where a contact is closed are pulled high and read as logic ones. Open return lines are pulled to ground by one of the 40 kΩ resistors and are read as zeroes. (The resistor values must be chosen to ensure all return lines are pulled above the 2.0V logic threshold, even in the worst-case,

where all contacts in an enabled column are closed.) Since P0 is provided open-collector outputs and high-impedance MOS inputs its input loading may be considered negligible.

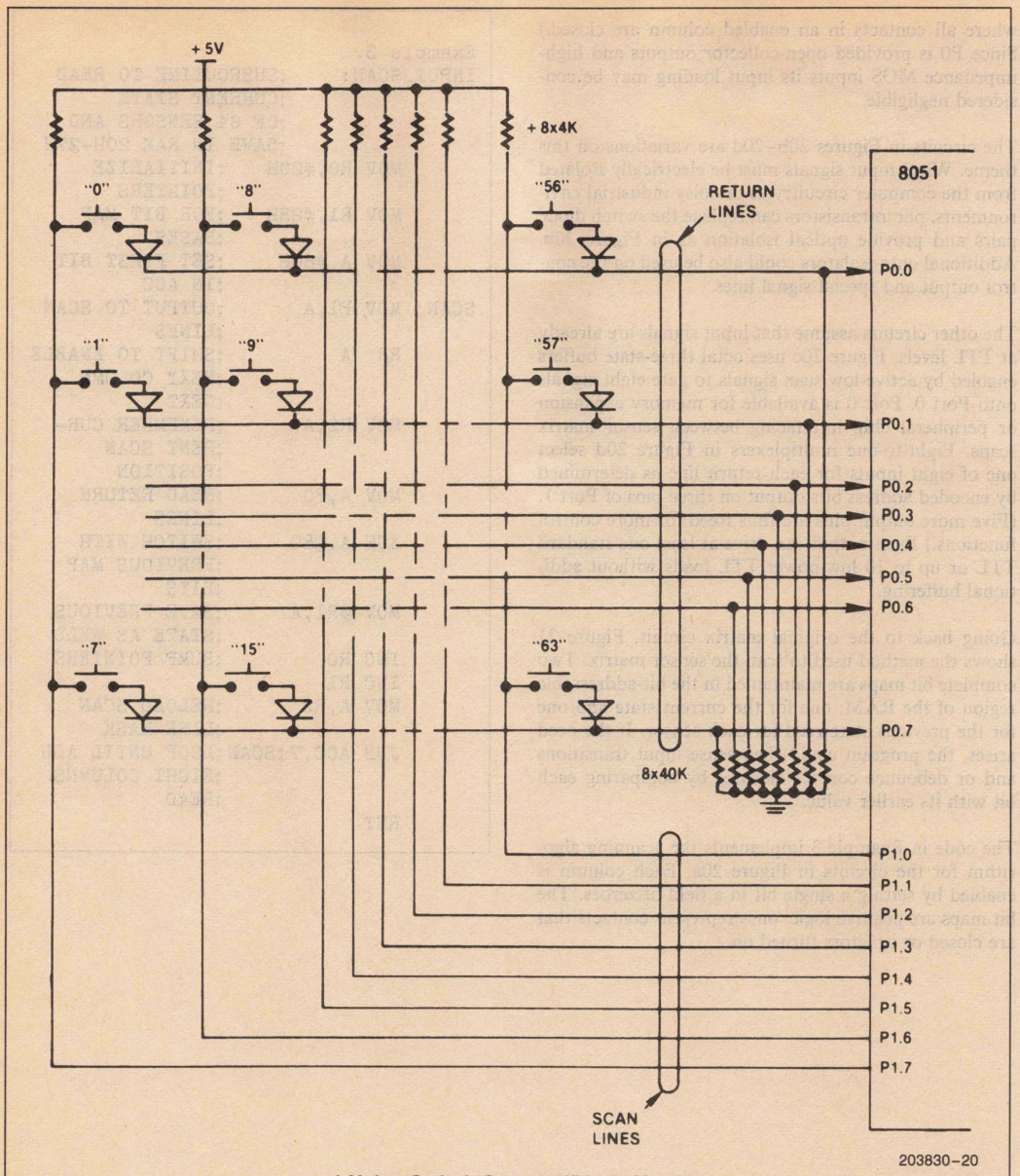
The circuits in Figures 20b–20d are variations on this theme. When input signals must be electrically isolated from the computer circuitry as in noisy industrial environments, phototransistors can replace the switch diode pairs and provide optical isolation as in Figure 20b. Additional opto-isolators could also be used on the control output and special signal lines.

The other circuits assume that input signals are already at TTL levels. Figure 20c uses octal three-state buffers enabled by active-low scan signals to gate eight signals onto Port 0. Port 0 is available for memory expansion or peripheral chip interfacing between sensor matrix scans. Eight-to-one multiplexers in Figure 20d select one of eight inputs for each return line as determined by encoded address bits output on three pins of Port 1. (Five more output pins are thus freed for more control functions.) Each output can drive at least one standard TTL or up to 10 low-power TTL loads without additional buffering.

Going back to the original matrix circuit, Figure 21 shows the method used to scan the sensor matrix. Two complete bit maps are maintained in the bit-addressable region of the RAM: one for the current state and one for the previous state read for each sensor. If the need arises, the program could then sense input transitions and or debounce contact closures by comparing each bit with its earlier value.

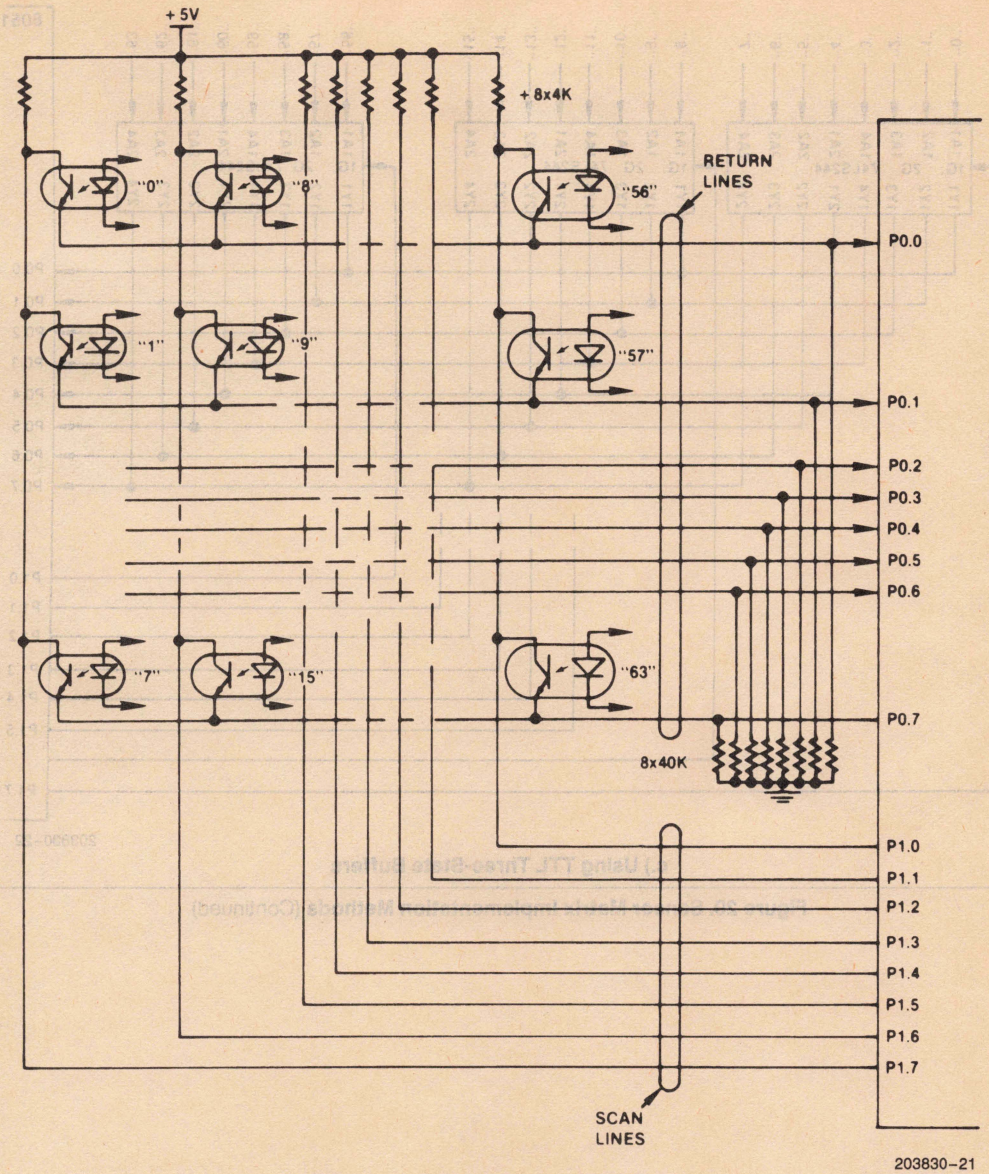
The code in Example 3 implements the scanning algorithm for the circuits in Figure 20a. Each column is enabled by setting a single bit in a field of zeroes. The bit maps are positive logic: ones represent contacts that are closed or isolators turned on.

```
Example 3.
INPUT_SCAN:     ;SUBROUTINE TO READ
                ;CURRENT STATE
                ;OF 64 SENSORS AND
                ;SAVE IN RAM 20H-27H
       MOV R0,#20H    ;INITIALIZE
                      ;POINTERS
       MOV R1,#28H    ;FOR BIT MAP
                      ;BASES
       MOV A,#80H     ;SET FIRST BIT
                      ;IN ACC
SCAN;  MOV P1,A       ;OUTPUT TO SCAN
                      ;LINES
       RR  A          ;SHIFT TO ENABLE
                      ;NEXT COLUMN
                      ;NEXT
       MOV R2,A       ;REMEMBER CUR-
                      ;RENT SCAN
                      ;POSITION
       MOV A,P0       ;READ RETURN
                      ;LINES
       XCH A,@R0      ;SWITCH WITH
                      ;PREVIOUS MAP
                      ;BITS
       MOV @R1,A      ;SAVE PREVIOUS
                      ;STATE AS WELL
       INC R0         ;BUMP POINTERS
       INC R1
       MOV A,R2       ;RELOAD SCAN
                      ;LINE MASK
       JNB ACC,7;SCAN;LOOP UNTIL ALL
                      ;EIGHT COLUMNS
                      ;READ
       RET
```



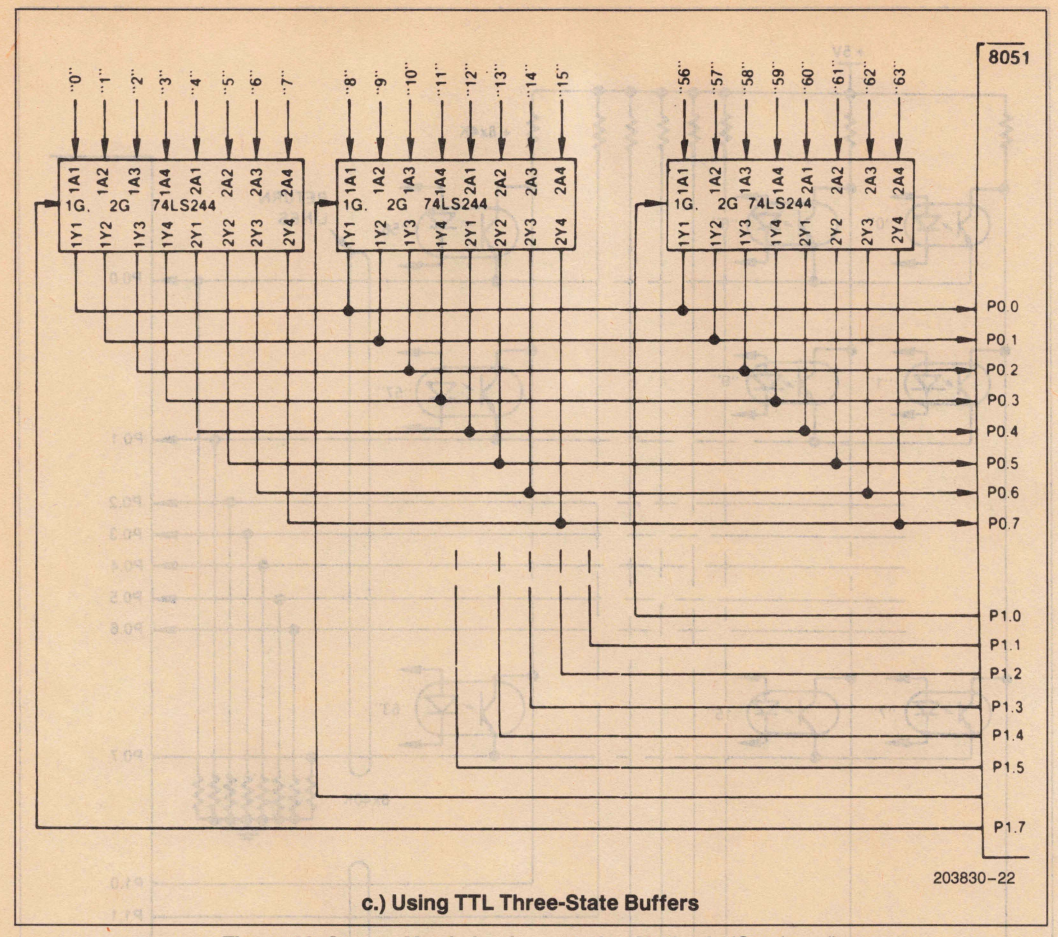
a.) Using Switch Contact/Diode Matrix

Figure 20. Sensor Matrix Implementation Methods



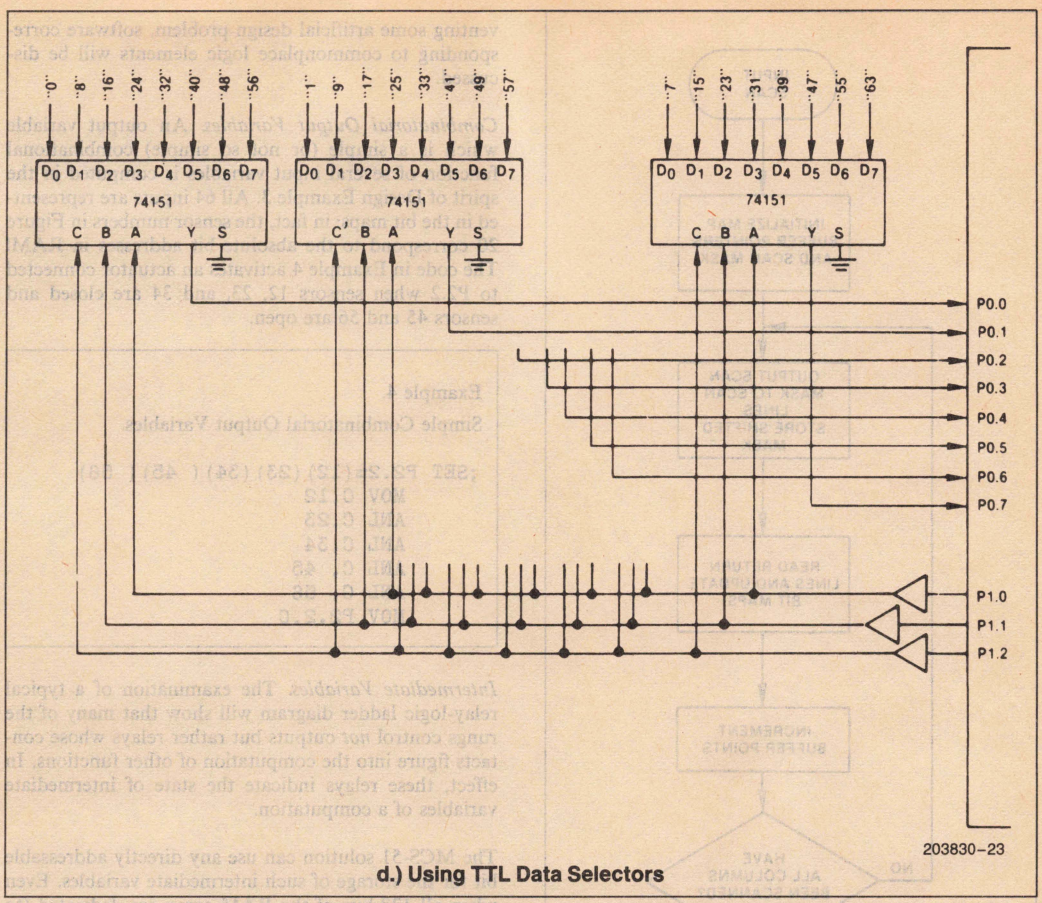
b.) Using Optically-Coupled Isolators

Figure 20. Sensor Matrix Implementation Methods (Continued)



c.) Using TTL Three-State Buffers

Figure 20. Sensor Matrix Implementation Methods (Continued)



d.) Using TTL Data Selectors

Figure 20. Sensor Matrix Implementation Methods (Continued)

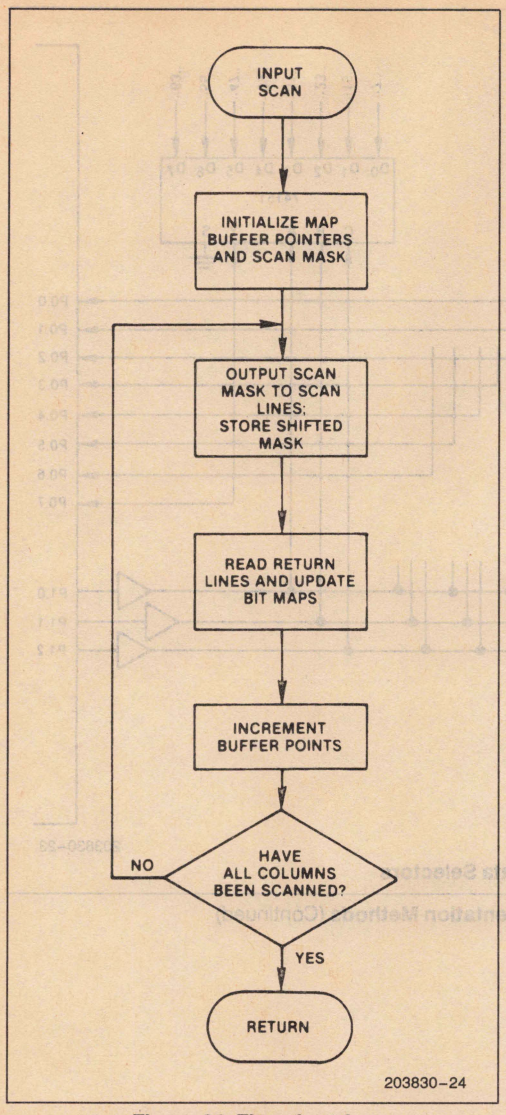


Figure 21. Flowchart for Reading in Sensor Matrix

What happens after the sensors have been scanned depends on the individual application. Rather than inventing some artificial design problem, software corresponding to commonplace logic elements will be discussed.

Combinatorial Output Variables. An output variable which is a simple (or not so simple) combinational function of several input variables is computed in the spirit of Design Example 3. All 64 inputs are represented in the bit maps: in fact, the sensor numbers in Figure 20 correspond to the absolute bit addresses in RAM! The code in Example 4 activates an actuator connected to P2.2 when sensors 12, 23, and 34 are closed and sensors 45 and 56 are open.

Example 4.
Simple Combinatorial Output Variables.

```
;SET P2.2=(12)(23)(34)( 45)( 56)
      MOV C,12
      ANL C,23
      ANL C,34
      ANL C, 45
      ANL C, 56
      MOV P2.2,C
```

Intermediate Variables. The examination of a typical relay-logic ladder diagram will show that many of the rungs control *not* outputs but rather relays whose contacts figure into the computation of other functions. In effect, these relays indicate the state of intermediate variables of a computation.

The MCS-51 solution can use any directly addressable bit for the storage of such intermediate variables. Even when all 128 bits of the RAM array are dedicated (to input bit maps in this example), the accumulator, PSW, and B register provide 18 additional flags for intermediate variables.

For example, suppose switches 0 through 3 control a safety interlock system. Closing any of them should deactivate certain outputs. Figure 22 is a ladder diagram for this situation. The interlock function could be recomputed for every output affected, or it may be computed once and save (as implied by the diagram). As the program proceeds this bit can qualify each output.

Example 5. Incorporating Override signal into actuator outputs.

```
;       CALL INPUT_SCAN
        MOV C,0
        ORL C,1
        ORL C,2
        ORL C,3
        MOV F0,C
;     .....  .....
;     COMPUTE FUNCTION 0
;
        ANL C, F0
        MOV PL0,C
;     .....  .....
;     COMPUTE FUNCTION 1
;
        ANL C, F0
        MOV P1,1,C
;     .....  .....
;     COMPUTE FUNCTION 2
;
        ANL C, F0
        MOV P1,2,C
;     .....  .....
```

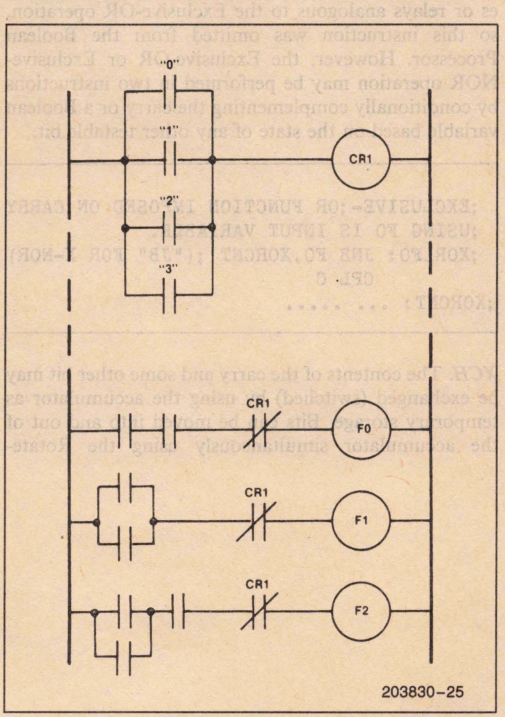


Figure 22. Ladder Diagram for Output Override Circuitry

Latching Relays. A latching relay can be forced into either the ON or OFF state by two corresponding input signals, where it will remain until forced onto the opposite state—analogous to a TTL Set/Reset flip-flop. The relay is used as an intermediate variable for other calculations. In the previous example, the emergency condition could be remembered and remain active until an "emergency cleared" button is pressed.

Any flag or addressable bit may represent a latching relay with a few lines of code (see Example 6).

Example 6. Simulating a latching relay.

```
;I_SET SET FLAG 0 IF C=1
I_SET:  ORL C,F0
        MOV F0,C
;
;I_RSET RESET FLAG 0 IF C=1
I_RSET: CPS C
        ANL C,F0
        MOV F0,C
;
```

Time Delay Relays. A time delay relay does not respond to an input signal until it has been present (or absent) for some predefined time. For example, a ballast or load resistor may be switched in series with a D.C. motor when it is first turned on, and shunted from the circuit after one second. This sort of time delay may be simulated by an interrupt routine driven by one of the two 8051 timer counters. The procedure followed by the routine depends heavily on the details of the exact function needed: time-outs or time delays with resettable or non-resettable inputs are possible. If the interrupt routine is executed every 10 milliseconds the code in Example 7 will clear an intermediate variable set by the background program after it has been active for two seconds.

Example 7. Code to clear USRFLG after a fixed time delay.

```
        JNB   USR_FLG,NXTTST
        DJNZ  DLAY_COUNT,NXTTST
        CLR   USR_FLG
        MOV   DLAY_COUNT,#200
NXTTST; ;..   .....
```

Serial Interface to Remote Processor. When it detects emergency conditions represented by certain input combinations (such as the earlier Emergency Override), the controller could shut down the machine immediately and/or alert the host processor via the serial port. Code bytes indicating the nature of the problem could be transmitted to a central computer. In fact, at 17,000 bytes-per-second, the entire contents of both bit maps could be sent to the host processor for further analysis in less than a millisecond! If the host decides that conditions warrant, it could alert other remote processors in the system that a problem exists and specify which shut-down sequence each should initiate. For more information on using the serial port, consult the MCS-51 User's Manual.

Response Timing

One difference between relay and programmed industrial controllers (when each is considered as a "black box") is their respective reaction times to input changes. As reflected by a ladder diagram, relay systems contain a large number of "rungs" operating in parallel. A change in input conditions will begin propagating through the system immediately, possibly affecting the output state within milliseconds.

Software, on the other hand, operates sequentially. A change in input states will not be detected until the next time an input scan is performed, and will not affect the outputs until that section of the program is reached. For that reason the raw speed of computing the logical functions is of extreme importance.

Here the Boolean processor pays off. *Every instruction mentioned in this Note* completes in one or two microseconds—the *minimum* instruction execution time for many other microcontrollers! A ladder diagram containing a hundred rungs, with an average of four contacts per rung can be replaced by approximately five hundred lines of software. A complete pass through the entire matrix scanning routine and all computations would require about a millisecond: less than the time it takes for most relays to change state.

A programmed controller which simulates each Boolean function with a subroutine would be less efficient by at least an order of magnitude. Extra software is needed for the simulation routines, and each step takes longer to execute for three reasons: several byte-wide logical instructions are executed per user program step (rather than one Boolean operation): most of those instructions take longer to execute with microprocessors performing multiple off-chip accesses: and calling and returning from the various subroutines requires overhead for stack operations.

In fact, the speed of the Boolean Processor solution is likely to be much faster than the system requires. The CPU might use the time left over to compute feedback parameters, collect and analyze execution statistics, perform system diagnostics, and so forth.

Additional Functions and Uses

With the building-block basics mentioned above many more operations may be synthesized by short instruction sequences.

Exclusive-OR. There are no common mechanical devices or relays analogous to the Exclusive-OR operation, so this instruction was omitted from the Boolean Processor. However, the Exclusive-OR or Exclusive-NOR operation may be performed in two instructions by conditionally complementing the carry or a Boolean variable based on the state of any other testable bit.

```
;EXCLUSIVE-;OR FUNCTION IMPOSED ON CARRY
;USING F0 IS INPUT VARIABLE.
;XOR_F0: JNB F0,XORCNT ;("JB" FOR X-NOR)
        CPL C
;XORCNT: ... .....
```

XCH. The contents of the carry and some other bit may be exchanged (switched) by using the accumulator as temporary storage. Bits can be moved into and out of the accumulator simultaneously using the Rotate-

through-carry instructions, though this would alter the accumulator data.

```
;EXCHANGE CARRY WITH USRFLG
XCHBIT: RLC    A
        MOV    C,USR_FLG
        RRC    A
        MOV    USR_FLG,C
        RLC    A
```

Extended Bit Addressing. The 8051 can directly address 144 general-purpose bits for all instructions in Figure 3b. Similar operations may be extended to any bit anywhere on the chip with some loss of efficiency.

The logical operations AND, OR, and Exclusive-OR are performed on byte variables using six different addressing modes, one of which lets the source be an immediate mask, and the destination any directly addressable byte. Any bit may thus be set, cleared, or complemented with a three-byte, two-cycle instruction if the mask has all bits but one set or cleared.

Byte variables, registers, and indirectly addressed RAM may be moved to a bit addressable register (usually the accumulator) in one instruction. Once transferred, the bits may be tested with a conditional jump, allowing any bit to be polled in 3 microseconds—still much faster than most architectures—or used for logical calculations. (This technique can also simulate additional bit addressing modes with byte operations.)

Parity of bytes or bits. The parity of the current accumulator contents is always available in the PSW, from whence it may be moved to the carry and further processed. Error-correcting Hamming codes and similar applications require computing parity on groups of isolated bits. This can be done by conditionally complementing the carry flag based on those bits or by gathering the bits into the accumulator (as shown in the DES example) and then testing the parallel parity flag.

Multiple byte shift and CRC codes

Though the 8051 serial port can accommodate eight- or nine-bit data transmissions, some protocols involve much longer bit streams. The algorithms presented in Design Example 2 can be extended quite readily to 16 or more bits by using multi-byte input and output buffers.

Many mass data storage peripherals and serial communications protocols include Cyclic Redundancy (CRC) codes to verify data integrity. The function is generally computed serially by hardware using shift registers and Exclusive-OR gates, but it can be done with software. As each bit is received into the carry, appropriate bits in the multi-byte data buffer are conditionally complemented based on the incoming data bit. When finished, the CRC register contents may be checked for zero by ORing the two bytes in the accumulator.

4.0 SUMMARY

A truly unique facet of the Intel MCS-51 microcomputer family design is the collection of features optimized for the one-bit operations so often desired in real-world, real-time control applications. Included are 17 special instructions, a Boolean accumulator, implicit and direct addressing modes, program and mass data storage, and many I/O options. These are the world's first single-chip microcomputers able to efficiently manipulate, operate on, and transfer either bytes or individual bits as data.

This Application Note has detailed the information needed by a microcomputer system designer to make full use of these capabilities. Five design examples were used to contrast the solutions allowed by the 8051 and those required by previous architectures. Depending on the individual application, the 8051 solution will be easier to design, more reliable to implement, debug, and verify, use less program memory, and run up to an order of magnitude faster than the same function implemented on previous digital computer architectures.

Combining byte- and bit-handling capabilities in a single microcomputer has a strong synergistic effect: the power of the result exceeds the power of byte- and bit-processors laboring individually. Virtually all user applications will benefit in some way from this duality. Data intensive applications will use bit addressing for test pin monitoring or program control flags: control applications will use byte manipulation for parallel I/O expansion or arithmetic calculations.

It is hoped that these design examples give the reader an appreciation of these unique features and suggest ways to exploit them in his or her own application.

APPENDIX A
Automobile Turn-Indicator Controller Program Listing

```
ISIS-II MCS-51 MACRO ASSEMBLER V1.0
OBJECT MODULE PLACED IN :F0:AP70.HEX
ASSEMBLER INVOKED BY:  :f1:asm51 ap70 src date(328)

LOC  OBJ             LINE     SOURCE

                       1      $XREF TITLE(AP-70 APPENDIX)
                       2      ;*******************************************************
                       3      ;
                       4      ;       THE FOLLOWING PROGRAM USES THE BOOLEAN INSTRUCTION SET
                       5      ;       OF THE INTEL 8051 MICROCOMPUTER TO PERFORM A NUMBER OF
                       6      ;       AUTOMOTIVE DASHBOARD CONTROL FUNCTIONS RELATING TO
                       7      ;       TURN SIGNAL CONTROL, EMERGENCY BLINKERS, BRAKE LIGHT
                       8      ;       CONTROL, AND PARKING LIGHT OPERATION.
                       9      ;       THE ALGORITHMS AND HARDWARE ARE DESCRIBED IN DESIGN
                      10      ;       EXAMPLE #4 OF INTEL APPLICATION NOTE AP-70,
                      11      ;                "USING THE INTEL MCS-51(TM)
                      12      ;              BOOLEAN PROCESSING CAPABILITIES"
                      13      ;
                      14      ;*******************************************************
                      15      ;
                      16      ;       INPUT PIN DECLARATIONS:
                      17      ;        (ALL INPUTS ARE POSITIVE-TRUE LOGIC
                      18      ;       INPUTS ARE HIGH WHEN RESPECTIVE SWITCH CONTACT IS CLOSED )
                      19      ;
0090                  20      BRAKE   BIT     P1.0    ; BRAKE PEDAL DEPRESSED
0091                  21      EMERG   BIT     P1.1    ; EMERGENCY BLINKER ACTIVATED
0092                  22      PARK    BIT     P1.2    ; PARKING LIGHTS ON
0093                  23      L_TURN  BIT     P1.3    ; TURN LEVER DOWN
0094                  24      R_TURN  BIT     P1.4    ; TURN LEVER UP
                      25      ;
                      26      ;       OUTPUT PIN DECLARATIONS:
                      27      ;       (ALL OUTPUTS ARE POSITIVE TRUE LOGIC
                      28      ;       BULB IS TURNED ON WHEN OUTPUT PIN IS HIGH.)
                      29      ;
0095                  30      L_FRNT  BIT     P1.5    ; FRONT LEFT-TURN INDICATOR
0096                  31      R_FRNT  BIT     P1.6    ; FRONT RIGHT-TURN INDICATOR
0097                  32      L_DASH  BIT     P1.7    ; DASHBOARD LEFT-TURN INDICATOR
00A0                  33      R_DASH  BIT     P2.0    ; DASHBOARD RIGHT-TURN INDICATOR
00A1                  34      L_REAR  BIT     P2.1    ; REAR LEFT-TURN INDICATOR
00A2                  35      R_REAR  BIT     P2.2    ; REAR RIGHT-TURN INDICATOR
                      36      ;
00A3                  37      S_FAIL  BIT     P2.3    ; ELECTRICAL SYSTEM FAULT INDICATOR
                      38      ;
                      39      ;       INTERNAL VARIABLE DEFINITIONS:
                      40      ;
0020                  41      SUB_DIV DATA    20H             ; INTERRUPT RATE SUBDIVIDER
0000                  42      HI_FREQ BIT     SUB_DIV.0       ; HIGH-FREQUENCY OSCILLATOR BIT
0007                  43      LO_FREQ BIT     SUB_DIV.7       ; LOW-FREQUENCY OSCILLATOR BIT
                      44      ;       ...     .....
00D1                  45      DIM     BIT     PSW.1           ; PARKING LIGHTS ON FLAG
                      46      ;
                      47      ;=======================================================
                      48 +1   $EJECT
```

203830-26

```
LOC  OBJ              LINE     SOURCE

                        49             ORG     0000H           ; RESET VECTOR
0000 020040             50             LJMP    INIT
                        51     ;
000B                    52             ORG     000BH           ; TIMER 0 SERVICE VECTOR
000B 758CF0             53             MOV     TH0,#-16        ; HIGH TIMER BYTE ADJUSTED TO CONTROL INT. RATE
000E C0D0               54             PUSH    PSW             ; EXECUTE CODE TO SAVE ANY REGISTERS USED BELOW
0010 0154               55             AJMP    UPDATE          ;     (CONTINUE WITH REST OF ROUTINE)
                        56     ;
0040                    57             ORG     0040H
0040 758A00             58     INIT:   MOV     TL0,#0          ; ZERO LOADED INTO LOW-ORDER BYTE AND
0043 758CF0             59             MOV     TH0,#-16        ; -16 IN HIGH-ORDER BYTE GIVES 4 MSEC PERIOD
0046 758961             60             MOV     TMOD,#01100001B ; 8-BIT AUTO RELOAD COUNTER MODE FOR TIMER 1,
                        61                                     ;  16-BIT TIMER MODE FOR TIMER 0 SELECTED
0049 7520F4             62             MOV     SUB_DIV,#244    ; SUBDIVIDE INTERRUPT RATE BY 244 FOR 1 HZ
004C D2A9               63             SETB    ET0             ; USE TIMER 0 OVERFLOWS TO INTERRUPT PROGRAM
004E D2AF               64             SETB    EA              ; CONFIGURE IE TO GLOBALLY ENABLE INTERRUPTS
0050 D28C               65             SETB    TR0             ; KEEP INSTRUCTION CYCLE COUNT UNTIL OVERFLOW
0052 80FE               66             SJMP    $               ; START BACKGROUND PROGRAM EXECUTION
                        67     ;
                        68     ;
0054 D52038             69     UPDATE: DJNZ    SUB_DIV,TOSERV  ; EXECUTE SYSTEM TEST ONLY ONCE PER SECOND
0057 7520F4             70             MOV     SUB_DIV,#244    ; GET VALUE FOR NEXT ONE SECOND DELAY AND
                        71     ;                               ; GO THROUGH ELECTRICAL SYSTEM TEST CODE:
005A 4390E0             72             ORL     P1,#11100000B   ; SET CONTROL OUTPUTS HIGH
005D 43A007             73             ORL     P2,#00000111B
0060 C295               74             CLR     L_FRNT          ; FLOAT DRIVE COLLECTOR
0062 20B428             75             JB      T0,FAULT        ; TO SHOULD BE PULLED LOW
0065 D295               76             SETB    L_FRNT          ; PULL COLLECTOR BACK DOWN
0067 C297               77             CLR     L_DASH          ; REPEAT SEQUENCE FOR L_DASH,
0069 20B421             78             JB      T0,FAULT
006C D297               79             SETB    L_DASH
006E C2A1               80             CLR     L_REAR          ;                 L_REAR,
0070 20B41A             81             JB      T0,FAULT
0073 D2A1               82             SETB    L_REAR
0075 C296               83             CLR     R_FRNT          ;                 R_FRNT,
0077 20B413             84             JB      T0,FAULT
007A D296               85             SETB    R_FRNT
007C C2A0               86             CLR     R_DASH          ;                 R_DASH,
007E 20B40C             87             JB      T0,FAULT
0081 D2A0               88             SETB    R_DASH
0083 C2A2               89             CLR     R_REAR          ;                 AND R_REAR
0085 20B405             90             JB      T0,FAULT
0088 D2A2               91             SETB    R_REAR
                        92     ;
                        93     ;       WITH ALL COLLECTORS GROUNDED, T0 SHOULD BE HIGH
                        94     ;       IF SO, CONTINUE WITH INTERRUPT ROUTINE.
                        95     ;
008A 20B402             96             JB      T0,TOSERV
008D B2A3               97     FAULT:  CPL     S_FAIL          ; ELECTRICAL FAILURE PROCESSING ROUTINE
                        98                                     ; (TOGGLE INDICATOR ONCE PER SECOND)
                        99 +1  $EJECT
```

203830-27

```
LOC  OBJ            LINE     SOURCE

                     100      ;       CONTINUE WITH INTERRUPT PROCESSING:
                     101      ;
                     102      ;   1)  COMPUTE LOW BULB INTENSITY WHEN PARKING LIGHTS ARE ON.
                     103      ;
008F A201            104      TOSERV: MOV     C,SUB_DIV 1     ; START WITH 50 PERCENT,
0091 8200            105              ANL     C,SUB_DIV 0     ; MASK DOWN TO 25 PERCENT,
0093 7202            106              ORL     C,SUB_DIV 2     ; BUILD BACK TO 62.5 PERCENT,
0095 8292            107              ANL     C,PARK          ; GATE WITH PARKING LIGHT SWITCH,
0097 92D1            108              MOV     DIM,C           ; AND SAVE IN TEMP. VARIABLE.
                     109      ;
                     110      ;   2)  COMPUTE AND OUTPUT LEFT-HAND DASHBOARD INDICATOR.
                     111      ;
0099 A293            112              MOV     C,L_TURN        ; SET CARRY IF TURN
009B 7291            113              ORL     C,EMERG         ; OR EMERGENCY SELECTED.
009D 8207            114              ANL     C,LO_FREQ       ; IF SO, GATE IN 1 HZ SIGNAL
009F 9297            115              MOV     L_DASH,C        ; AND OUTPUT TO DASHBOARD.
                     116      ;
                     117      ;   3)  COMPUTE AND OUTPUT LEFT-HAND FRONT TURN SIGNAL.
                     118      ;
00A1 92D5            119              MOV     F0,C            ; SAVE FUNCTION SO FAR.
00A3 72D1            120              ORL     C,DIM           ; ADD IN PARKING LIGHT FUNCTION
00A5 9295            121              MOV     L_FRNT,C        ; AND OUTPUT TO TURN SIGNAL.
                     122      ;
                     123      ;   4)  COMPUTE AND OUTPUT LEFT-HAND REAR TURN SIGNAL.
                     124      ;
00A7 A290            125              MOV     C,BRAKE         ; GATE BRAKE PEDAL SWITCH
00A9 B093            126              ANL     C,/L_TURN       ; WITH TURN LEVER.
00AB 72D5            127              ORL     C,F0            ; INCLUDE TEMP. VARIABLE FROM DASH
00AD 72D1            128              ORL     C,DIM           ; AND PARKING LIGHT FUNCTION
00AF 92A1            129              MOV     L_REAR,C        ; AND OUTPUT TO TURN SIGNAL.
                     130      ;
                     131      ;   5)  REPEAT ALL OF ABOVE FOR RIGHT-HAND COUNTERPARTS.
                     132      ;
00B1 A294            133              MOV     C,R_TURN        ; SET CARRY IF TURN
00B3 7291            134              ORL     C,EMERG         ; OR EMERGENCY SELECTED.
00B5 8207            135              ANL     C,LO_FREQ       ; IF SO, GATE IN 1 HZ SIGNAL
00B7 92A0            136              MOV     R_DASH,C        ; AND OUTPUT TO DASHBOARD.
00B9 92D5            137              MOV     F0,C            ; SAVE FUNCTION SO FAR.
00BB 72D1            138              ORL     C,DIM           ; ADD IN PARKING LIGHT FUNCTION
00BD 9296            139              MOV     R_FRNT,C        ; AND OUTPUT TO TURN SIGNAL.
00BF A290            140              MOV     C,BRAKE         ; GATE BRAKE PEDAL SWITCH
00C1 B094            141              ANL     C,/R_TURN       ; WITH TURN LEVER.
00C3 72D5            142              ORL     C,F0            ; INCLUDE TEMP. VARIABLE FROM DASH
00C5 72D1            143              ORL     C,DIM           ; AND PARKING LIGHT FUNCTION
00C7 92A2            144              MOV     R_REAR,C        ; AND OUTPUT TO TURN SIGNAL.
                     145      ;
                     146      ;       RESTORE STATUS REGISTER AND RETURN.
                     147      ;
00C9 D0D0            148              POP     PSW             ; RESTORE PSW
00CB 32              149              RETI                    ; AND RETURN FROM INTERRUPT ROUTINE
                     150      ;
                     151              END
```

203830-28

```
XREF SYMBOL TABLE LISTING
---- ------ ----- -------


NAME      TYPE     VALUE AND REFERENCES

BRAKE     N BSEG   0090H  20# 125 140
DIM       N BSEG   00D1H  45# 108 120 128 138 143
EA        N BSEG   00AFH  64
EMERG     N BSEG   0091H  21# 113 134
ETO       N BSEG   00A9H  63
FO        N BSEG   00D5H  119 127 137 142
FAULT     L CSEG   008DH  75 78 81 84 87 90 97#
HI_FREQ   N BSEG   0000H  42#
INIT      L CSEG   0040H  50 58#
L_DASH    N BSEG   0097H  32# 77 79 115
L_FRNT    N BSEG   0095H  30# 74 76 121
L_REAR    N BSEG   00A1H  34# 80 82 129
L_TURN    N BSEG   0093H  23# 112 126
LO_FREQ   N BSEG   0007H  43# 114 135
P1        N DSEG   0090H  20 21 22 23 24 30 31 32 72
P2        N DSEG   00A0H  33 34 35 37 73
PARK      N BSEG   0092H  22# 107
PSW       N DSEG   00D0H  45 54 148
R_DASH    N BSEG   00A0H  33# 86 88 136
R_FRNT    N BSEG   0096H  31# 83 85 139
R_REAR    N BSEG   00A2H  35# 89 91 144
R_TURN    N BSEG   0094H  24# 133 141
S_FAIL    N BSEG   00A3H  37# 97
SUB_DIV   N DSEG   0020H  41# 42 43 62 69 70 104 105 106
TO        N BSEG   00B4H  75 78 81 84 87 90 96
TOSERV    L CSEG   008FH  69 96 104#
TH0       N DSEG   008CH  53 59
TL0       N DSEG   008AH  58
TMOD      N DSEG   0089H  60
TR0       N BSEG   008CH  65
UPDATE    L CSEG   0054H  55 69#


ASSEMBLY COMPLETE, NO ERRORS FOUND
```

203830-29

intel

APPLICATION NOTE

AP-252

August 1986

Designing With The 80C51BH

TOM WILLIAMSON
MCO APPLICATIONS ENGINEER

Order Number: 270068-002

CMOS EVOLVES

The original CMOS logic families were the 4000-series and the 74C-series circuits. The 74C-series circuits are functional equivalents to the corresponding numbered 74-series TTL circuits, but have CMOS logic levels and retain the other well known characteristics of CMOS logic.

These characteristics are: low power consumption, high noise immunity, and slow speed. The low power consumption is inherent to the nature of the CMOS circuit. The noise immunity is due partly to the CMOS logic levels, and partly to the slowness of the circuits. The slow speed is due to the technology used to construct the transistors in the circuit.

The technology used is called metal-gate CMOS, because the transistor gates are formed by metal deposition. More importantly, the gates are formed after the drain and source regions have been defined, and must overlap the source and drain somewhat to allow for alignment tolerances. This overlap plus the relatively large size of the transistors themselves result in high electrode capacitance, and that is what limits the speed of the circuit.

High speed CMOS became feasible with the development of the self-aligning silicon gate technology. In this process polysilicon gates are deposited **before** the source and drain regions are defined. Then the source and drain regions are formed by ion implantation using the gate itself as a mask for the implantation. This eliminates most of the overlap capacitance. In addition, the process allows smaller transistors. The result is a significant increase in circuit speed. The 74HC-series of CMOS logic circuits is based on this technology, and has speeds comparable to LS TTL, which is to say about 10 times faster than the 74C-series circuits.

The size reduction that contributes to the higher speed also demands an accompanying reduction in the maximum supply voltage. High-speed CMOS is generally limited to 6V.

WHAT IS CHMOS?

CHMOS is the name given to Intel's high-speed CMOS processes. There are two CHMOS processes, one based on an n-well structure and one based on a p-well structure. In the n-well structure, n-type wells are diffused into a p-type substrate. Then the n-channel transistors (nFETs) are built into the substrate and pFETs are built into the n-wells. In the p-well structure, p-type wells are diffused into an n-type substrate. Then the nFETs are built into the wells and pFETs, into the substrate. Both processes have their advantages and disadvantages, which are largely transparent to the user.

Lower operating voltages are easier to obtain with the p-well structure than with the n-well structure. But the p-well structure does not easily adapt to an EPROM which would be pin-for-pin compatible with HMOS EPROMs. On the other hand the n-well structure can be based on the solidly founded HMOS process, in which nFETs are built into a p-type substrate. This allows somewhat more than half of the transistors in a CHMOS chip to be constructed by processes that are already well characterized.

Currently Intel's CHMOS microcontrollers and memory products are n-well devices, whereas CHMOS microprocessors are p-well devices.

Further discussion of the CHMOS technology is provided in References 1 and 2 (which are reprinted in the Microcontroller Handbook).

THE MCS®-51 FAMILY IN CHMOS

The 80C51BH is the CHMOS version of Intel's original 8051. The 80C31BH is the ROMless 80C51BH, equivalent to the 8031. These CHMOS devices are architecturally identical with their HMOS counterparts, except that they have two added features for reduced power. These are the Idle and Power Down modes of operation.

In most cases, an 80C51BH can directly replace the 8051 in existing applications. It can execute the same code at the same speed, accept signals from the same sources, and drive the same loads. However, the 80C51BH covers a wider range of speeds, will emit CMOS logic levels to CMOS loads, and will draw about 1/10 the current of an 8051 (and less yet in the reduced power modes). Interchangeability between the HMOS and CHMOS devices is discussed in more detail in the final section of this Application Note.

It should be noted that the 80C51BH CPU is not static. That means if the clock frequency is too low, the CPU might forget what it was doing. This is because the circuitry uses a number of dynamic nodes. A dynamic node is one that uses the note-to-ground capacitance to form a temporary storage cell. Dynamic nodes are used to reduce the transistor count, and hence the chip area, thus to produce a more economical device.

This is not to say that the on-chip RAM in CHMOS microcontrollers is dynamic. It's not. It's the CPU that is dynamic, and that is what imposes the minimum clock frequency specification.

LATCHUP

Latchup is an SCR-type turn-on phenomenon that is the traditional nemesis of CMOS systems. The substrate, the wells, and the transistors form parasitic pnpn structures within the device. These parasitic structures turn on like an SCR if a sufficient amount of forward current is driven through one of the junctions. From the circuit designer's point of view it can happen whenever an input or output pin is externally driven a diode drop above V_{CC} or below V_{SS}, by a source that is capable of supplying the required trigger current.

However much of a problem latchup has been in the past, it is good to know that in most recently developed CMOS devices, and specifically in CHMOS devices, the current required to trigger latchup is typically well over 100 mA. The 80C51BH is virtually immune to latchup. (References 1 and 2 present a discussion of the latchup mechanisms and the steps that are taken on the chip to guard against it.) Modern CMOS is not absolutely immune to latchup, but with trigger currents in the hundreds of mA, latchup is certainly a lot easier to avoid than it once was.

A careless power-up sequence might trigger a latchup in the older CMOS families, but it's unlikely to be a major problem in high-speed CMOS or in CHMOS. There is still some risk incurred in inserting or removing chips or boards in a CMOS system while the power is on. Also, severe transients, such as inductive kicks or momentary short-circuits, can exceed the trigger current for latchup.

For applications in which some latchup risk seems unavoidable, you can put a small resistor (100Ω or so) in series with signal lines to ensure that the trigger current will never be reached. This also helps to control overshoot and RFI.

LOGIC LEVELS AND INTERFACING PROBLEMS

CMOS logic levels differ from TTL levels in two ways. First, for equal supply voltages, CMOS gives (and requires) a higher "logic 1" level than TTL. Secondly, CMOS logic levels are V_{CC} (or VDD) dependent, whereas guaranteed TTL logic levels are fixed when V_{CC} is within TTL specs.

Standard 74HC logic levels are as follows:

$V_{IH}MIN = 70\%$ of V_{CC}
$V_{IL}MAX = 20\%$ of V_{CC}
$V_{OH}MIN = V_{CC} - 0.1V$, $|I_{OH}| \leq 20\ \mu A$
$V_{OL}MAX = 0.1V$, $|I_{OL}| \leq 20\ \mu A$

Figure 1 compares 74HC, LS TTL, and 74HCT logic levels with those of the HMOS 8051 and the CHMOS 80C51BH for $V_{CC} = 5V$.

Output logic levels depend of course on load current, and are normally specified at several load currents. When CMOS and TTL are powered by the same V_{CC}, the logic levels guaranteed on the data sheets indicate that CMOS can drive TTL, but TTL can't drive CMOS. The incompatibility is that the TTL circuit's V_{OH} level is too low to reliably be recognized by the CMOS circuit as a valid V_{IH}.

Since HMOS circuits were designed to be TTL-compatible, they have the same incompatibility.

Fortunately, 74HCT-series circuits are available to ease these interfacing problems. They have TTL-compatible logic levels at the inputs and standard CMOS levels at the outputs.

The 80C51BH is designed to work with either TTL or CMOS. Therefore its logic levels are specified very much like 74HCT circuits. That is, its input logic levels are TTL-compatible, and its output characteristics are like standard high-speed CMOS.

NOISE CONSIDERATIONS

One of the major reasons for going to CMOS has traditionally been that CMOS is less susceptible to noise. As previously noted, its low susceptibility to noise is

Logic State	$V_{CC} = 5V$				
	74HC	74HCT	LS TTL	8051	80C51BH
V_{IH}	3.5V	2.0V	2.0V	2.0V	1.9V
V_{IL}	1.0V	0.8V	0.8V	0.8V	0.9V
V_{OH}	4.9V	4.9V	2.7V	2.4V	4.5V
V_{OL}	0.1V	0.1V	0.5V	0.45V	0.45V

Figure 1. Logic Level Comparison. (Output voltage levels depend on load current. Data sheets list guaranteed output levels for several load currents. The output levels listed here are for minimum loading.)

partly due to superior noise margins, and partly due to its slow speed.

Noise margin is the difference between V_{OL} and V_{IL}, or between V_{OH} and V_{IH}. If V_{OH} from a driving circuit is 2.7V and V_{IH} to the driven circuit is 2.0V, then the driven circuit has 0.7V of noise margin at the logic high level. These kinds of comparisons show that an all-CMOS system has wider noise margins than an all-TTL system.

Figure 2 shows noise margins in CMOS and LS TTL systems when both have V_{CC} = 5V. It can be seen that CMOS/CMOS and CMOS/CHMOS systems have an edge over LS TTL in this respect.

Noise margins can be misleading, however, because they don't say how much noise energy it takes to induce in the circuit a noise voltage of sufficient amplitude to cause a logic error. This would involve consideration of the width of the noise pulse as compared with the circuit's response speed, and the impedance to ground from the point of noise introduction in the circuit.

When these considerations are included, it is seen that using the slower 74C- and 4000-series circuits with a 12 or 15V supply voltage does offer a truly improved level of noise immunity, but that high-speed CMOS at 5V is not significantly better than TTL.

One should not mistake the wider supply voltage tolerance of high-speed CMOS for V_{CC} glitch immunity. Supply voltage tolerance is a DC rating, not a glitch rating.

For any clocked CMOS, and most especially for VLSI CMOS, V_{CC} decoupling is critical. CHMOS draws current in extremely sharp spikes at the clock edges. The VHF and UHF components of these spikes are not drawn from the power supply, but from the decoupling capacitor. If the decoupling circuit is not sufficiently low in inductance, V_{CC} will glitch at each clock edge. We suggest that a 0.1 μF decoupler cap be used in a minimum-inductance configuration with the microcontroller. A minimum-inductance configuration is one that minimizes the area of the loop formed by the chip (V_{CC} to V_{SS}), the traces to the decoupler cap, and the decoupler cap. PCB designers too often fail to understand that if the traces that connect the decoupler cap to the V_{CC} and V_{SS} pins aren't short and direct, the decoupler loses much of its effectiveness.

Interface	Noise Margin for V_{CC} = 5V	
	Logic Low $V_{IL}-V_{OL}$	Logic High $V_{OH}-V_{IH}$
74HC to 74HC	0.9V	1.4V
LSTTL to LSTTL	0.3V	0.7V
LSTTL to 74HCT	0.3V	0.7V
LSTTL to 80C51BH	0.3V	0.7V
74HC to 80C51BH	0.8V	3.0V
80C51BH to 74HC	0.8V	1.0V

Figure 2. Noise Margins for CMOS and LS TTL Circuits

Overshoot and ringing in signal lines are potential sources of logic upsets. These can largely be controlled by circuit layout. Inserting small resistors (about 100Ω) in series with signal lines that seem to need them will also help.

The sharp edges produced by high-speed CMOS can cause RFI problems. The severity of these problems is largely a function of the PCB layout. We don't mean to imply that all RFI problems can be solved by a better PCB layout. It may well be, for example, that in some RFI-sensitive designs high-speed CMOS is simply not the answer. But circuit layout is a critical factor in the noise performance of any electronic system, and more so in high-speed CMOS systems than others.

Circuit layout techniques for minimizing noise susceptibility and generation are discussed in References 3 through 6.

UNUSED PINS

CMOS input pins should not be left to float, but should always be pulled to one logic level or the other. If they float, they tend to float into the transition region between 0 and 1, where the pullup and pulldown devices in the input buffer are both conductive. This causes a significant increase in I_{CC}. A similar effect exists in HMOS circuits, but with less noticeable results.

In 80C51BH and 80C31BH designs, unused pins of Ports 1, 2, and 3 can be ignored, because they have internal pullups that will hold them at a valid Logic 1 level. Port 0 pins are different, however, in not having internal pullups (except during bus operations).

When the 80C51BH is in reset, the Port 0 pins are in a float state unless they are externally pulled up or down. If it's going to be held in reset for just a short time, the transient float state can probably be ignored. When it comes out of reset, the pins stay afloat unless

they are externally pulled either up or down. Alternatively, the software can internally write 0s to whatever Port 0 pins may be unused.

The same considerations are applicable to the 80C31BH with regards to reset. But when the 80C31BH comes out of reset, it commences bus operations, during which the logic levels at the pins are always well defined as high or low.

Consider the 80C31BH in the Power Down or Idle modes, however. In those modes it is not fetching instructions, and the Port 0 pins will float if not externally pulled high or low. The choice of whether to pull them high or low is the designer's. Normally it is sufficient to pull them up to V_{CC} with 10k resistors. But if power is going to be removed from circuits that are connected to the bus, it will be advisable to pull the bus pins down (normally with 10k resistors). Considerations involved in selecting pullup and pulldown resistor values are as follows.

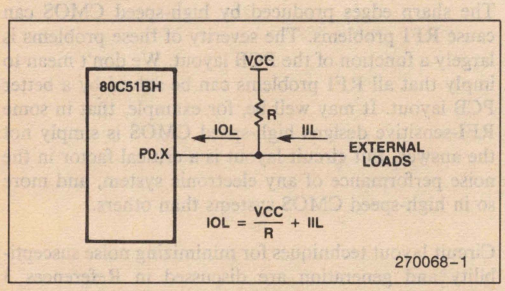


Figure 3a. Conditions defining the minimum value for R. P0.X is emitting a logic low. R must be large enough to not cause IOL to exceed data sheet specifications.

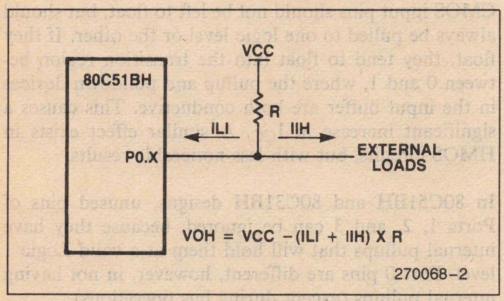


Figure 3b. Conditions defining the maximum value for R. P0.X is in a high impedance state. R must be small enough to keep V_{OH} acceptably high.

PULLUP RESISTORS

If a pullup resistor is to be used on a Port 0 pin, its minimum value is determined by I_{OL} requirements. If the pin is trying to emit a 0, then it will have to sink the current from the pullup resistor plus whatever other current may be sourced by other loads connected to the pin, as shown in Figure 3a, while maintaining a valid output low (V_{OL}). To guarantee that the pin voltage will not exceed 0.45V, the resistor should be selected so that I_{OL} doesn't exceed the value specified on the data sheet. In most CMOS applications, the minimum value would be about 2k Ω.

The maximum value you could use depends on how fast you want the pin to pull up after bus operations have ceased, and how high you want the V_{OH} level to be. The smaller the resistor the faster it pulls up. Its effect on the V_{OH} level is that $V_{OH} = V_{CC} - (ILI + IIH) \times R$. ILI is the input leakage current to the Port 0 pin, and IIH is the input high current to the external loads, as shown in Figure 3b. Normally V_{OH} can be expected to reach 0.9 V_{CC} if the pullup resistance does not exceed about 50k Ω.

Pulldown Resistors

If a pulldown resistor is to be used on a Port 0 pin, its minimum value is determined by V_{OH} requirements during bus operations, and its maximum value is in most cases determined by leakage current.

During bus operations the port uses internal pullups to emit 1s. The D.C. Characteristics in the data sheet list guaranteed V_{OH} levels for given I_{OH} currents. (The "-" sign in the I_{OH} value means the pin is sourcing that current to the external load, as shown in Figure 4.) To ensure the V_{OH} level listed in the data sheet, the resis-

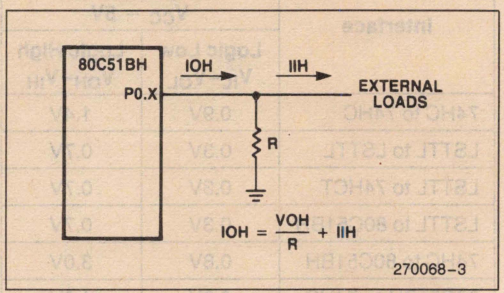


Figure 4a. Conditions defining the minimum value for R. P0.X is emitting a 1 in a bus operation. R must be large enough to not cause IOH to exceed data sheet specifications.

$$\frac{V_{OH}}{R} + I_{IH} \leq |I_{OH}|$$

tor has to satisfy where I_{IH} is the input high current to the external loads.

When the pin goes into a high impedance state, the pulldown resistor will have to sink leakage current from the pin, plus whatever other current may be sourced by other loads connected to the pin, as shown in Figure 4b. The Port 0 leakage current is I_{LI} on the data sheet. The resistor should be selected so that the voltage developed across it by these currents will be seen as a logic low by whatever circuits are connected to it (including the 80C51BH). In CMOS/CHMOS applications, 50k Ω is normally a reasonable maximum value.

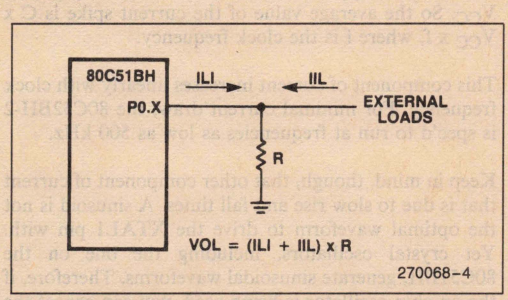


Figure 4b. Conditions defining the maximum value for R. P0.X is in a high impedance state. R must be small enough to keep VOL acceptably low.

DRIVE CAPABILITY OF THE INTERNAL PULLUPS

There's an important difference between HMOS and CHMOS port drivers. The pins of Ports 1, 2, and 3 of the CHMOS parts each have three pullups: strong, normal, and weak, as shown in Figure 5. The strong pullup (p1) is only used during 0-to-1 transitions, to hasten the transition. The weak pullup (p2) is on whenever the bit latch contains a 1. The "normal" pullup (p3) is controlled by the pin voltage itself.

The reason that p3 is controlled by the pin voltage is that if the pin is being used as an input, and the external source pulls it to a low, then turning off p3 makes for a lower I_{IL}. The data sheet shows an "I_{TL}" specification. This is the current that p3 will source during the time the pin voltage is making its 1-to-0 transition. This is what I_{IL} would be if an input low at the pin didn't turn p3 off.

Note, however, that this p3 turn-off mechanism puts a restriction on the drive capacity of the pin if it's being used as an output. If you're trying to output a logic high, and the external load pulls the pin voltage below the pin's V_{IH}MIN spec, p3 might turn off, leaving only the weak p2 to provide drive to the load. To prevent this happening, you need to ensure that the load doesn't draw more than the I_{OH} spec for a valid V_{OH}. The idea is to make sure the pin voltage never falls below its own V_{IH}MIN specification.

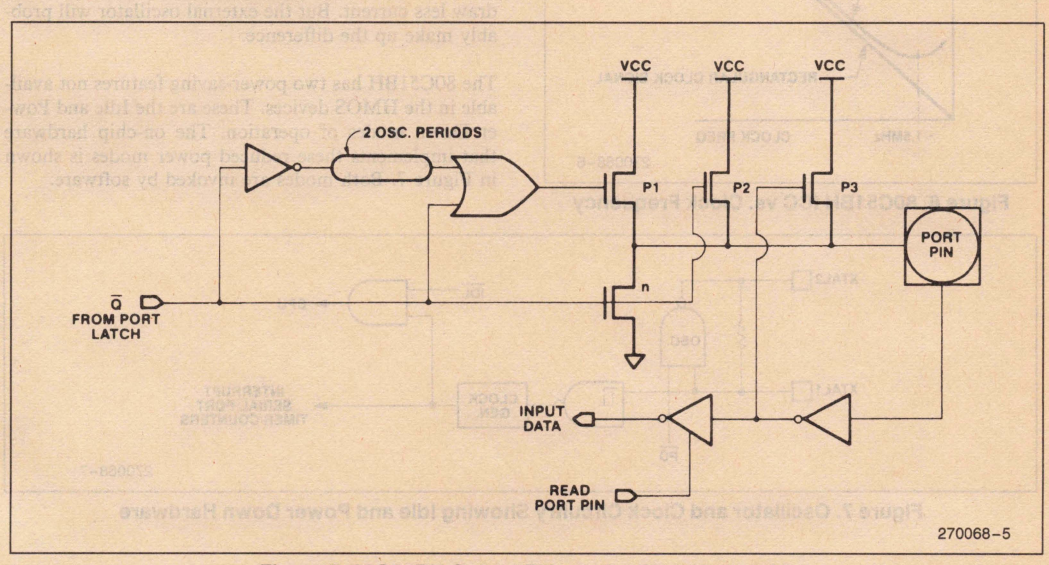


Figure 5. 80C51BH Output Drivers for Ports 1, 2 and 3

POWER CONSUMPTION

The main reason for going to CMOS, of course, is to conserve power. (There are other reasons, but this is the main one.) Conserving power doesn't mean just reducing your electric bill. Nor does it necessarily relate to battery operation, although battery operation without CMOS is pretty unhandy. The main reason for conserving power is to be able to put more functionality into a smaller space. The reduced power consumption allows the use of smaller and lighter power supplies, and less heat being generated allows denser packaging of circuit components. Expensive fans and blowers can usually be eliminated.

A cooler running chip is also more reliable, since most random and wearout failures relate to die temperature. And finally, the lower power dissipation will allow more functions to be integrated onto the chip.

The reason CMOS consumes less power than NMOS is that when it's in a stable state there is no path of conduction from V_{CC} to V_{SS} except through various leakage paths. CMOS does draw current when it's changing states. How much current it draws depends on how often and how quickly it changes states.

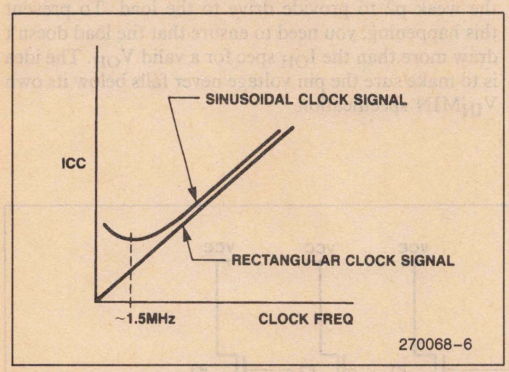


Figure 6. 80C51BH ICC vs. Clock Frequency

CMOS circuits draw current in sharp spikes during logical transitions. These current spikes are made up of two components. One is the current that flows during the transition time when pullup and pulldown FETs are both active. The average (DC) value of this component is larger when the transition times of the input signals are longer. For this reason, if the current draw is a critical factor in the design, slow rise and fall times should be avoided, even when the system speed doesn't seem to justify a need for nanosecond switching speeds.

The other component is the current that charges stray and load capacitance at the nodes of a CMOS logic gate. The average value of this current spike is its area (integral over time) multiplied by its rep rate. Its area is the amount of charge it takes to raise the node capacitance, C, to V_{CC}. That amount of charge is just C x V_{CC}. So the average value of the current spike is C x V_{CC} x f, where f is the clock frequency.

This component of current increases linearly with clock frequency. For minimal current draw, the 80C52BH-2 is spec'd to run at frequencies as low as 500 kHz.

Keep in mind, though, that other component of current that is due to slow rise and fall times. A sinusoid is not the optimal waveform to drive the XTAL1 pin with. Yet crystal oscillators, including the one on the 80C51BH, generate sinusoidal waveforms. Therefore, if the on-chip oscillator is being used, you can expect the device to draw more current at 500 kHz, than it does at 1.5 MHz, as shown in Figure 6. If you derive a good sharp square wave from an external oscillator, and use that to drive XTAL1, then the microcontroller will draw less current. But the external oscillator will probably make up the difference.

The 80C51BH has two power-saving features not available in the HMOS devices. These are the Idle and Power Down modes of operation. The on-chip hardware that implements these reduced power modes is shown in Figure 7. Both modes are invoked by software.

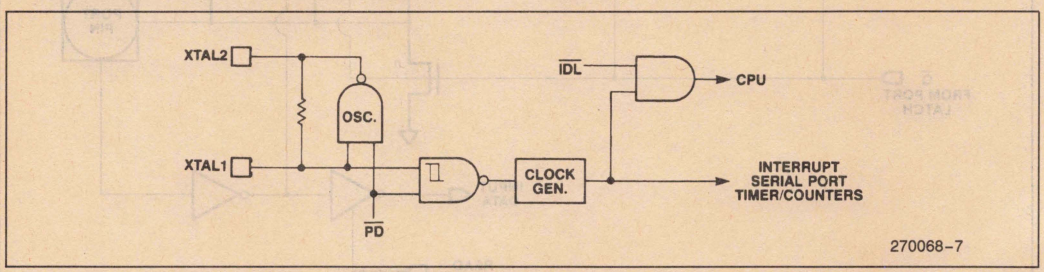


Figure 7. Oscillator and Clock Circuitry Showing Idle and Power Down Hardware

Idle: In the Idle Mode ($\overline{IDL}$ = 0 in Figure 7), the CPU puts itself to sleep by gating off its own clock. It doesn't stop the oscillator. It just stops the internal clock signal from getting to the CPU. Since the CPU draws 80 to 90 percent of the chip's power, shutting it off represents a fairly significant power savings. The on-chip peripherals (timers, serial port, interrupts, etc.) and RAM continue to function as normal. The CPU status is preserved in its entirety: the Stack Pointer, Program Counter, Program Status Word, Accumulator, and all other registers maintain their data during Idle.

The Idle Mode is invoked by setting bit 0 (IDL) of the PCON register. PCON is not bit-addressable, so the bit has to be set by a byte operation, such as

```
ORL PCON,#1
```

The PCON register also contains flag bits GF0 and GF1, which can be used for any general purposes, or to give an indication if an interrupt occurred during normal operation or during Idle. In this application, the instruction that invokes Idle also sets one or both of the flag bits. Their status can then be checked in the interrupt routines.

While the device is in the Idle Mode, ALE and $\overline{PSEN}$ emit logic high (V_{OH}), as shown in Figure 8. This is so external EPROM can be deselected and have its output disabled.

The port pins hold the logical states they had at the time the Idle was activated. If the device was executing out of external program memory, Port 0 is left in a high impedance state and Port 2 continues to emit the high byte of the program counter (using the strong pullups to emit 1s). If the device was executing out of internal program memory, Ports 0 and 2 continue to emit whatever is in the P0 and P2 registers.

There are two ways to terminate Idle. Activation of any enabled interrupt will cause the hardware to clear bit 0 of the PCON register, terminating the Idle mode. The interrupt will be serviced, and following RETI the next instruction to be executed will be the one following the instruction that invoked Idle.

The other way is with a hardware reset. Since the clock oscillator is still running, RST only needs to be held active for two machine cycles (24 oscillator periods) to complete the reset. Note that this exit from Idle writes 1s to all the ports, initializes all SFRs to their reset values, and restarts program execution from location 0.

Power Down: In the Power Down Mode ($\overline{PD}$ = 0 in Figure 7), the CPU puts the whole chip to sleep by turning off the oscillator. In case it was running from an external oscillator, it also gates off the path to the internal phase generators, so no internal clock is generated even if the external oscillator is still running. The on-chip RAM, however, saves its data, as long as V_{CC} is maintained. In this mode the only I_{CC} that flows is leakage, which is normally in the micro-amp range.

The Power Down Mode is invoked by setting bit 1 in the PCON register, using a byte instruction such as

```
ORL   PCON,#2
```

While the device is in Power Down, ALE and $\overline{PSEN}$ emit lows (V_{OL}), as shown in Figure 8. The reason they are designed to emit lows is so that power can be removed from the rest of the circuit, if desired, while the 80CS51BH is in its Power Down mode.

The port pins continue to emit whatever data was written to them. Note that Port 2 emits its P2 register data even if execution was from external program memory.

Pin	Internal Execution		External Execution	
	Idle	Power Down	Idle	Power Down
ALE	1	0	1	0
PSEN/	1	0	1	0
P0	SFR Data	SFR Data	High-Z	High-Z
P1	SFR Data	SFR Data	SFR Data	SFR Data
P2	SFR Data	SFR Data	PCH	SFR Data
P3	SFR Data	SFR Data	SFR Data	SFR Data

Figure 8. Status of Pins in Idle and Power Down Modes. "SFR data" means the port pins emit their internal register data. "PCH" is the high byte of the Program Counter.

Port 0 also emits its P0 register data, but if execution was from external program memory, the P0 register data is FF. The oscillator is stopped, and the part remains in this state as long as V_{CC} is held, and until it receives an external reset signal.

The only exit from Power Down is a hardware reset. Since the oscillator was stopped, RST must be held active long enough for the oscillator to re-start and stabilize. Then the reset function initializes all the Special Function Registers (ports, timers, etc.) to their reset values, and re-starts the program from location 0. Therefore, timer reloads, interrupt enables, baud rates, port status, etc. need to be re-established. Reset does not affect the content of the on-chip data RAM. If V_{CC} was held during Power Down, the RAM data is still good.

USING THE POWER DOWN MODE

The software-invoked Power Down feature offers a means of reducing the power consumption to a mere trickle in systems which are to remain dormant for some period of time, while retaining important data.

The user should give some thought to what state the port pins should be left in during the time the clock is stopped, and write those values to the port latches before invoking Power Down.

If V_{CC} is going to be held to the entire circuit, one would want to write values to the port latches that would deselect peripherals before invoking Power Down. For example, if external memory is being used, the P2 SFR should be loaded with a value which will not generate an active chip select to any memory device.

In some applications, V_{CC} to part of the system may be shut off during Power Down, so that even quiescent and standby currents are eliminated. Signal lines that connect to those chips must be brought to a logic low, whether the chip in question is CMOS, NMOS, or TTL, before V_{CC} is shut off to them. CMOS pins have parasitic pn junctions to V_{CC}, which will be forward biased if V_{CC} is reduced to zero while the pin is held at a logic high. NMOS pins often have FETs that look like diodes to V_{CC}. TTL circuits may actually be damaged by an input high if $V_{CC} = 0$. That's why the 80C51BH outputs lows at ALE and $\overline{PSEN}$ during Power Down.

Figure 9 shows a circuit that can be used to turn V_{CC} off to part of the system during Power Down. The circuit will ensure that the secondary circuit is not de-energized until after the 80C31BH is in Power Down, and that the 80C31BH does not receive a reset (terminating the Power Down mode) before the secondary circuit is re-energized. Therefore, the program memory itself can be part of the secondary circuit.

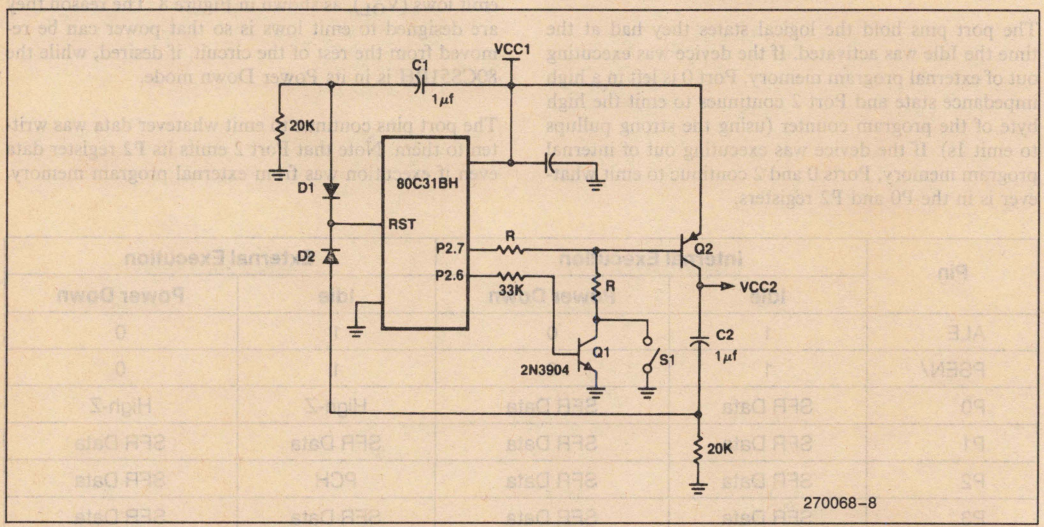


Figure 9. The 80C31BH de-energizes part of the circuit (VCC2) when it goes into Power Down. Selections of R and Q2 depend on VCC2 current draw.

In Figure 9, when V_{CC} is switched on to the 80C31BH, capacitor C1 provides a power-on reset. The reset function writes 1s to all the port pins. The 1 at P2.6 turns Q1 on, enabling V_{CC} to the secondary circuit through transistor Q2. As the 80C31BH comes out of reset, Port 2 commences emitting the high byte of the Program Counter, which results in the P2.7 and P2.6 pins outputting 0s. The 0 at P2.7 ensures continuation of V_{CC} to the secondary circuit.

The system software must now write a 1 to P2.7 and a 0 to P2.6 in the Port 2 SFR, P2. These values will not appear at the Port 2 pins as long as the device is fetching instructions from external program memory. However, whenever the 80C31BH goes into Power Down, these values will appear at the port pins, and will shut off both transistors, disabling V_{CC} to the secondary circuit.

Closing the switch S1 re-energizes the secondary circuit, and at the same time sends a reset through C2 to the 80C31BH to wake it up. The diode D1 is to prevent C1 from hogging current from C2 during this secondary reset. D2 prevents C2 from discharging through the RST pin when V_{CC} to the secondary circuit goes to zero.

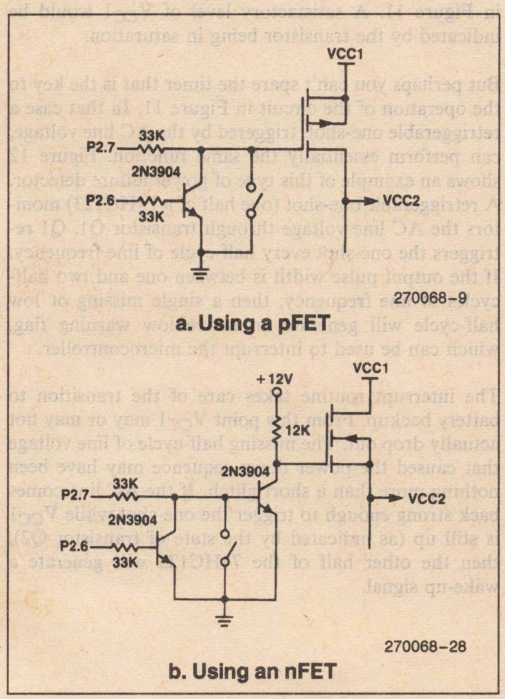


Figure 10. Using Power MOSFETs to Control VCC2

USING POWER MOSFETs to CONTROL V_{CC}

Power MOSFETs are gaining in popularity (and availability). The easiest way to control V_{CC} is with a Logic Level pFET, as shown in Figure 10a. This circuit allows the full V_{CC} to be used to turn the device on. Unfortunately, power pFETs are not economically competitive with bipolar transistors of comparable ratings.

Power nFETs are both economical and available, and can be used in this application if a DC supply of higher voltage is available to drive the gate. Figure 10b shows how to implement a V_{CC} switch using a power nFET and a (nominally) + 12V supply. The problem here is that if the device is on, its source voltage is +5V. To maintain the on state, the gate has to be another 5 or 10V above that. The "12V" supply is not particularly critical. A minimally filtered, unregulated rectifier will suffice.

BATTERY BACKUP SYSTEMS

Here we consider circuits that normally draw power from the AC line, but switch to battery operation in the event of a power failure. We assume that in battery operation high-current loads will be allowed to die along with the AC power. The system may continue then with reduced functionality, monitoring a control transducer, perhaps, or driving an LCD. Or it may go into a bare-bones survival mode, in which critical data is saved but nothing else happens till AC power is restored.

In any case it is necessary to have some early warning of an impending power failure so that the system can arrange an orderly transfer to battery power. Early warning systems can operate by monitoring either the AC line voltage or the unregulated rectifier output, or even by monitoring the regulated DC voltage.

Monitoring the AC line voltage gives the earliest warning. That way you can know within one or two half-cycles of line frequency that AC power is down. In most cases you then have at least another half-cycle of line frequency before the regulated V_{CC} starts to fall. In a half-cycle of line frequency an 80C51BH can execute about 5,000 instructions—plenty of time to arrange an orderly transfer of power.

The circuit in Figure 11 uses a Zener diode to test the line voltage each half-cycle, and a junction transistor to pass the information on to the 80C51BH. (Obviously a voltage comparator with a suitable reference source can

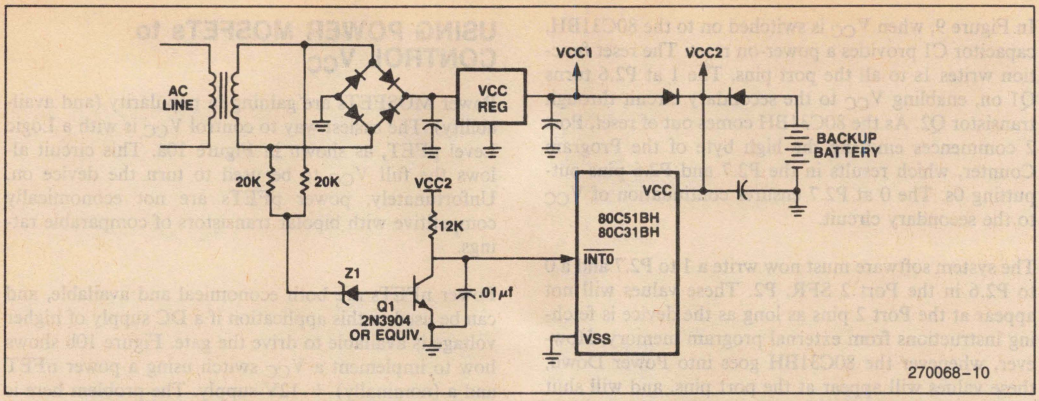


Figure 11. Power Failure Detector with Battery Backup. When AC power fails, VCC1 goes down and VCC2 is held.

perform the same function, if one prefers.) The way it works is if the line voltage reaches an acceptably high level, it breaks over Z1, drives Q1 to saturation, and interrupts the 80C51BH. The interrupt would be transition-activated, in this application. The interrupt service routine reloads one of the C51BH's timers to a value that will make it roll over in something between one and two half-cycles of line frequency. As long as the line voltage is healthy, the timer never rolls over, because it is reloaded every half-cycle. If there is a single half-cycle in which the line voltage doesn't reach a high enough level to generate the interrupt, the timer rolls over and generates a **timer** interrupt.

The timer interrupt then commences the transition to battery backup. Critical data needs to be copied into protected RAM. Signals to circuits that are going to lose power must be written to logic low. Protected circuits (those powered by $V_{CC}2$) that communicate with unprotected circuits must be deselected. The microcontroller itself may be put into Idle, so that it can continue some level of interrupt-driven functionality, or it may be put into Power Down.

Note that if the CPU is going to invoke Power Down, the Special Function Registers may also need to be copied into protected RAM, since the reset that terminates the Power Down mode will also intialize all the SFRs to their reset values.

The circuit in Figure 11 does not show a wake-up mechanism. A number of choices are available, however. A pushbutton could be used to generate an interrupt, if the CPU is in Idle, or to activate reset, if the CPU is in Power Down.

Automatic wake-up on power restoration is also possible. If the CPU is in Idle, it can continue to respond to any interrupts that might be generated by Q1. The interrupt service routine determines from the status of flag bits GF0 and GF1 in PCON that it is in Idle because there was a power outage. It can then sample $V_{CC}1$ through a voltage comparator similar to Z1, Q1 in Figure 11. A satisfactory level of $V_{CC}1$ would be indicated by the transistor being in saturation.

But perhaps you can't spare the timer that is the key to the operation of the circuit in Figure 11. In that case a retriggerable one-shot, triggered by the AC line voltage, can perform essentially the same function. Figure 12 shows an example of this type of power failure detector. A retriggerable one-shot (one half of a 74HC123) monitors the AC line voltage through transistor Q1. Q1 retriggers the one-shot every half-cycle of line frequency. If the output pulse width is between one and two half-cycles of line frequency, then a single missing or low half-cycle will generate an active low warning flag, which can be used to interrupt the microcontroller.

The interrupt routine takes care of the transition to battery backup. From this point $V_{CC}1$ may or may not actually drop out. The missing half-cycle of line voltage that caused the power down sequence may have been nothing more than a short glitch. If the AC line comes back strong enough to trigger the one-shot while $V_{CC}1$ is still up (as indicated by the state of transistor Q2), then the other half of the 74HC123 will generate a wake-up signal.

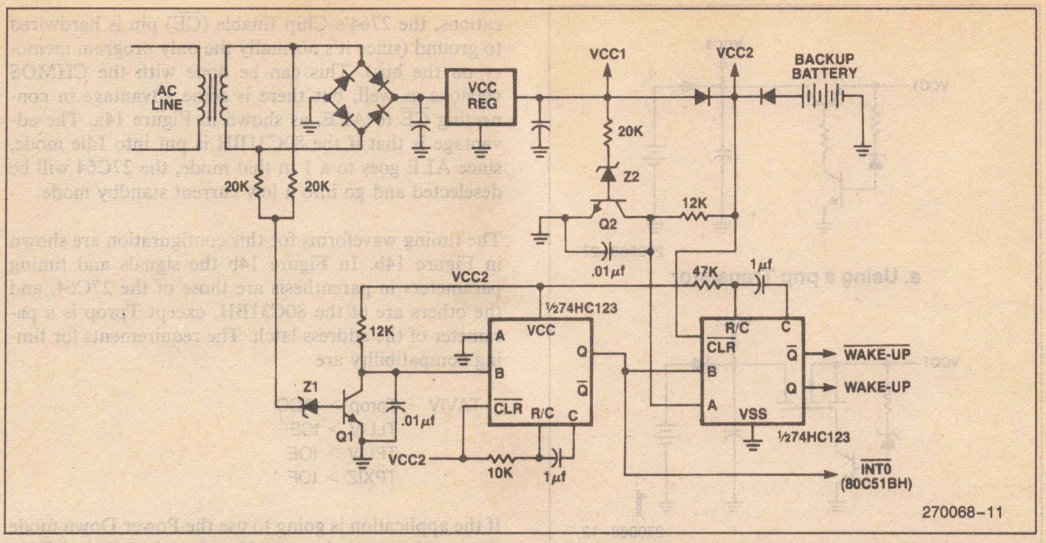


Figure 12. Power Failure Detector uses retriggerable one-shots to flag impending power outage and generate automatic wake-up when power returns.

Having been awakened, the 80C51BH will stay awake for at least another half-cycle of line frequency (another 5,000 or so instructions) before possibly being told to arrange another transfer of power. Consequently, if the line voltage is jittering erratically around the switchover point (determined by diode Z1), the system will limp along executing in half-cycle units of line frequency.

On the other hand, if the power outage is real and lengthy, $V_{CC}1$ will eventually fall below the level at which the backup battery takes over. The backup battery maintains power to the 80C51BH, and to the 74HC123, and to whatever other circuits are being protected during this outage. The battery voltage must be high enough to maintain $V_{CC}MIN$ specs to the 80C51BH.

If the microcontroller is an 80C31BH, executing out of external ROM, and if the C31BH is put into Idle during the power outage, then the external ROM must also be supplied by the battery. On the other hand, if the C31BH is put into Power Down during the outage, then the ROM can be allowed to die with the AC power. The considerations here are the same as in Figure 9: V_{CC} to the ROM is still up at the time Power Down is invoked, and we must ensure (through selection of diode Z2 in Figure 12) that the 80C31BH is not awakened till ROM power is back in spec.

POWER SWITCHOVER CIRCUITS

Battery backup systems need to have a way for the protected circuits to draw power from the line-operated power supply when that source is available, and to switch over to battery power when required. The switchover circuit is simple if the entire system is to be battery powered in the event of a line power outage. In that case a pair of diodes suffice, as shown in Figure 12, provided $V_{CC}MIN$ specs are still met after the diode drop has been subtracted from its respective power source.

The situation becomes more complicated when part of the circuit is going to be allowed to die when the AC power goes out. In that case it is difficult to maintain equal V_{CC}s to protected and unprotected circuits (and possibly dangerous not to).

The problem can be alleviated by using a Schottky diode instead of a 1N4001, for its lower forward voltage drop. The 1N5820, for example, has a foward drop of about 0.35V at 1A.

Other solutions are to use a transistor or power MOSFET switch, as shown in Figure 13. With minor modifications this switch can be controlled by port pins.

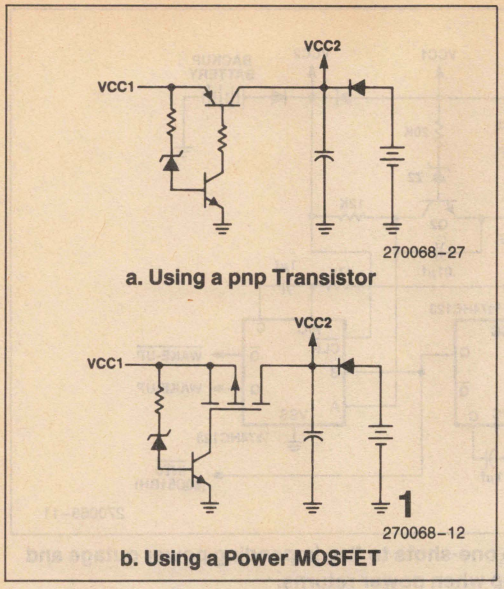


Figure 13. Power Switchover Ckts.

80C31BH + CHMOS EPROM

The 27C64 and 87C64 are Intel's 8K byte CHMOS EPROMs. The 27C64 requires an external address latch, and can be used with the 80C31BH as shown in Figure 14a. In most 8031 + 2764 (HMOS) applications, the 2764's Chip Enable ($\overline{CE}$) pin is hardwired to ground (since it's normally the only program memory on the bus). This can be done with the CHMOS versions as well, but there is some advantage in connecting $\overline{CE}$ to ALE, as shown in Figure 14a. The advantage is that if the 80C31BH is put into Idle mode, since ALE goes to a 1 in that mode, the 27C64 will be deselected and go into a low current standby mode.

The timing waveforms for this configuration are shown in Figure 14b. In Figure 14b the signals and timing parameters in parenthesis are those of the 27C64, and the others are of the 80C31BH, except Tprop is a parameter of the address latch. The requirements for timing compatibility are

TAVIV − Tprop > tACC
TLLIV > tCE
TPLIV > tOE
TPXIZ > tDF

If the application is going to use the Power Down mode then we have another consideration: In Idle, ALE = $\overline{PSEN}$ = 1, and in Power Down, ALE = $\overline{PSEN}$ = 0. In a realistic application there are likely to be more chips in the circuit than are shown in Figure 14, and it is likely that the nonessential ones will have their V_{CC} removed while the CPU is in Power Down. In that case the EPROM and the address latch should be among the chips that have V_{CC} removed, and logic lows are exactly what are required at ALE and $\overline{PSEN}$.

But if V_{CC} is going to be maintained to the EPROM during Power Down, then it will be necessary to de-

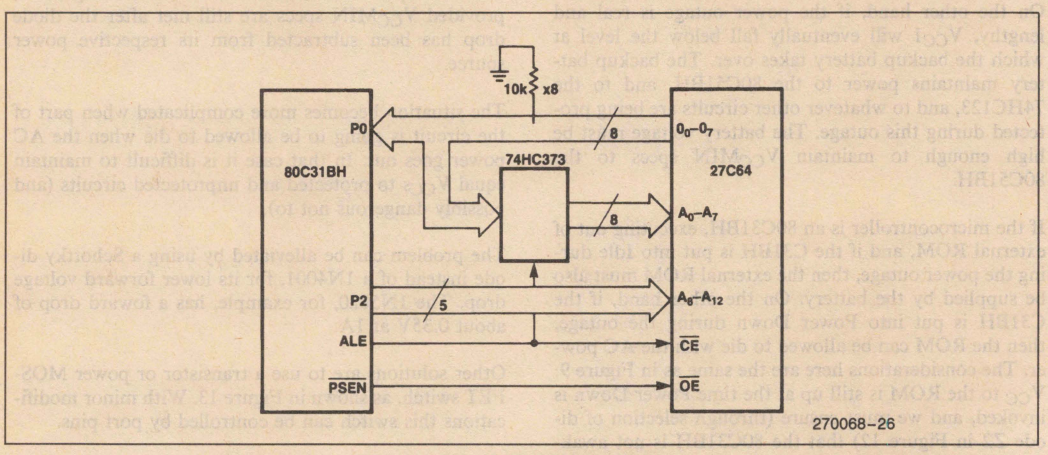


Figure 14a. 80C31BH + 27C64

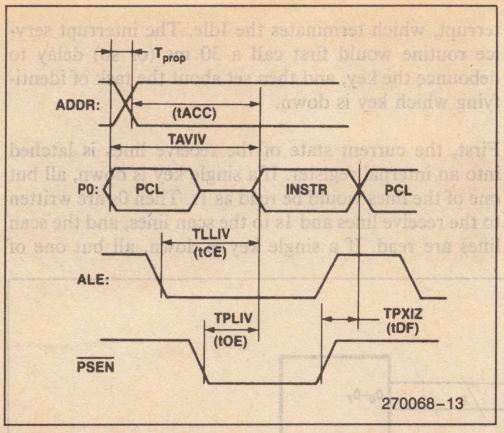


Figure 14b. Timing Waveforms for 80C31BH + 27C64

select the EPROM when the CPU is in Power Down. If Idle is never invoked, $\overline{CE}$ of the EPROM can be connected to P2.7 of the 80C31BH, as shown in Figure 15a. In normal operation, P2.7 will be emitting the MSB of the Program Counter, which is 0 if the program contains less than 32K of code. Then when the CPU goes into Power Down, the Port 2 pins emit P2 SFR data, which puts a 1 at P2.7, thus deselecting the EPROM.

If Idle and Power Down are both going to be used, $\overline{CE}$ of the EPROM can be driven by the logical OR of ALE and P2.7, as shown in Figure 15b. In Idle, ALE = 1 will deselect the EPROM, and in Power Down, P2.7 = 1 will deselect it.

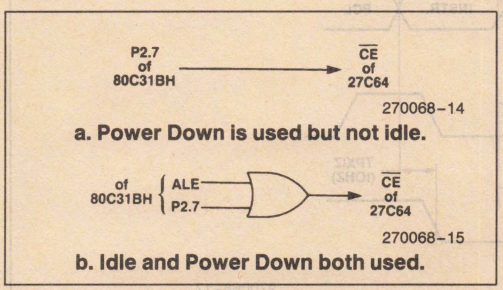


Figure 15. Modifications to 80C31BH/27C64 Interface

Pulldown resistors are shown in Figure 14a under the assumption that something on the bus is going to have its V_{CC} removed during Power Down. If this is not the case, pullups can be used as well as pulldowns.

The 87C64 is like the 27C64 except that it has an on-chip address latch. The Port 0 pins are tied to both address and data pins of the 87C64, as shown in Figure 16a. ALE drives the EPROM's ALE/$\overline{CS}$ input. During ALE high, the address information is allowed to flow into the EPROM and begin accessing the code byte. On the falling edge of ALE the address byte is internally latched. The A0–A7 inputs are then ignored and the same bus lines are used to transmit the fetched code byte from the O0–O7 pins back to the 80C31BH.

The timing waveforms for this configuration are shown in Figure 16b. In Figure 16b the signals and timing parameters in parentheses are those of the 87C64, and the others are of the 80C31BH. The requirements for timing compatibility are

TLHLL > tLL

TAVLL > tAL

TLLAX > tLA

TLLIV > tACL

TPLIV > tOE

TLLPL > tCOE

TPXIZ > tOHZ

The same considerations apply to the 87C64 as to the 27C64 with regards to the Idle and Power Down modes. Basically you want $\overline{CS}$ = 1 if V_{CC} is maintained to the EPROM, and $\overline{CS}$ = $\overline{OE}$ = 0 if V_{CC} is removed.

SCANNING A KEYBOARD

There are many different kinds of keyboards, but alphanumeric keyboards generally consist of a matrix of 8 scan lines and 8 receive lines as shown in Figure 17. Each set of lines connects to one port of the microcontroller. The software has written 0s to the scan lines, and 1s to the receive lines. Pressing a key connects a scan line to a receive line, thus pulling the receive line to a logic low.

The 8 receive lines are ANDed to one of the external interrupt pins, so that pulling any of the receive lines low generates an interrupt. The interrupt service routine has to identify the pressed key, if only one key is down, and convert that information to some useful output. If more than one key in the line matrix is found to be pressed, no action is taken. (This is a "two key lockout" scheme.)

On some keyboards, certain keys (Shift, Control, Escape, etc.) are not a part of the line matrix. These keys would connect directly to a port pin on the microcontroller, and would not cause lock-out if pressed simultaneously with a matrix key, nor generate an interrupt if pressed singly.

Normally the microcontroller would be in idle mode when a key has not been pressed, and another task is not in progress. Pressing a matrix key generates an interrupt, which terminates the Idle. The interrupt service routine would first call a 30 ms (or so) delay to debounce the key, and then set about the task of identifying which key is down.

First, the current state of the receive lines is latched into an internal register. If a single key is down, all but one of the lines would be read as 1s. Then 0s are written to the receive lines and 1s to the scan lines, and the scan lines are read. If a single key is down, all but one of

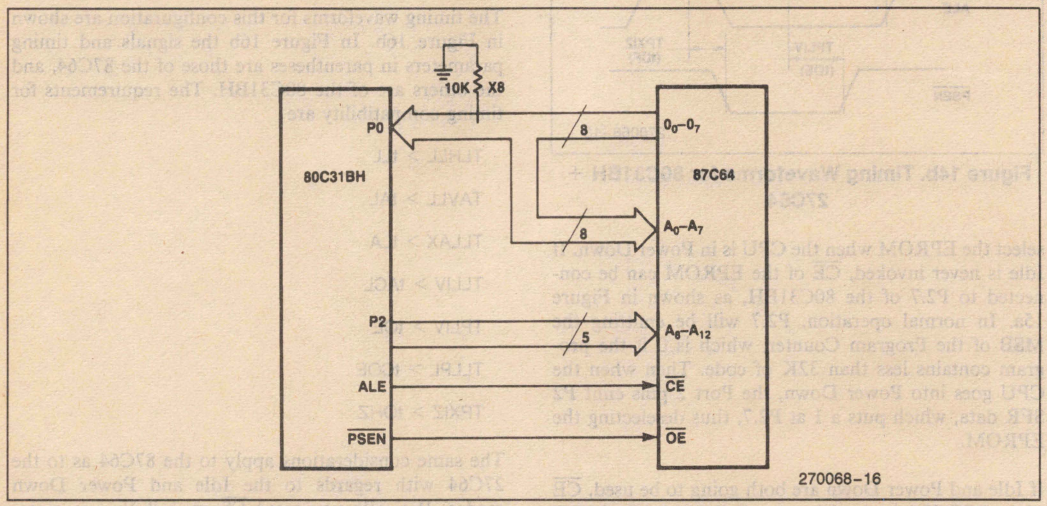


Figure 16a. 80C31BH + 87C64

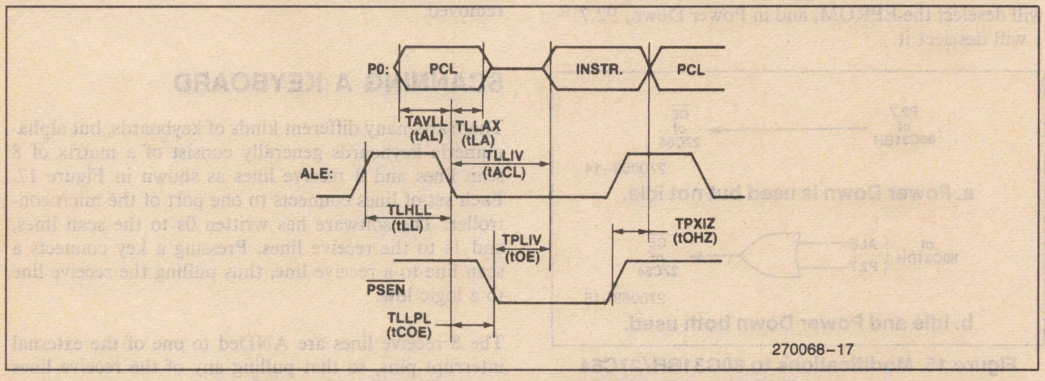


Figure 16b. Timing Waveforms for 80C31BH + 87C64

these lines would be read as 1s. By locating the single 0 in each set of lines, the pressed key can be identified. If more than one matrix key is down, one or both sets of lines will contain multiple 0s.

A subroutine is used to determine which of 8 bits in either set of lines is 0, and whether more than one bit is 0. Figure 18 shows a subroutine (SCAN) which does that using the 8051's bit-addressing capability. To use the subroutine, move the line data into a bit-addressable RAM location named LINE, and call the SCAN routine. The number of LINE bits which are zero is returned in ZERO__COUNTER. If only one bit is zero, its number (1 through 8) is returned in ZERO__BIT.

The interrupt service routine that is executed in response to a key closure might then be as follows:

```
RESPONSE_TO_KEY_CLOSURE:
        CALL    DEBOUNCE_DELAY
        MOV     LINE,P1; ;See Figure 17.
        CALL    SCAN
        DJNZ    ZERO_COUNTER,REJECT
        MOV     ADDRESS,ZERO_BIT
        MOV     P2,#0FFH; ;See Figure 17.
        MOV     P1,#0
        MOV     LINE,P2
        CALL    SCAN
        DJNZ    ZERO_COUNTER,REJECT
        XCH     A,ZERO_BIT
        SWAP    A
        ORL     ADDRESS,A
        XCH     A,ZERO_BIT
        MOV     P1,#0FFH
        MOV     P2,#0
REJECT: CLR     EX0
        RETI
```

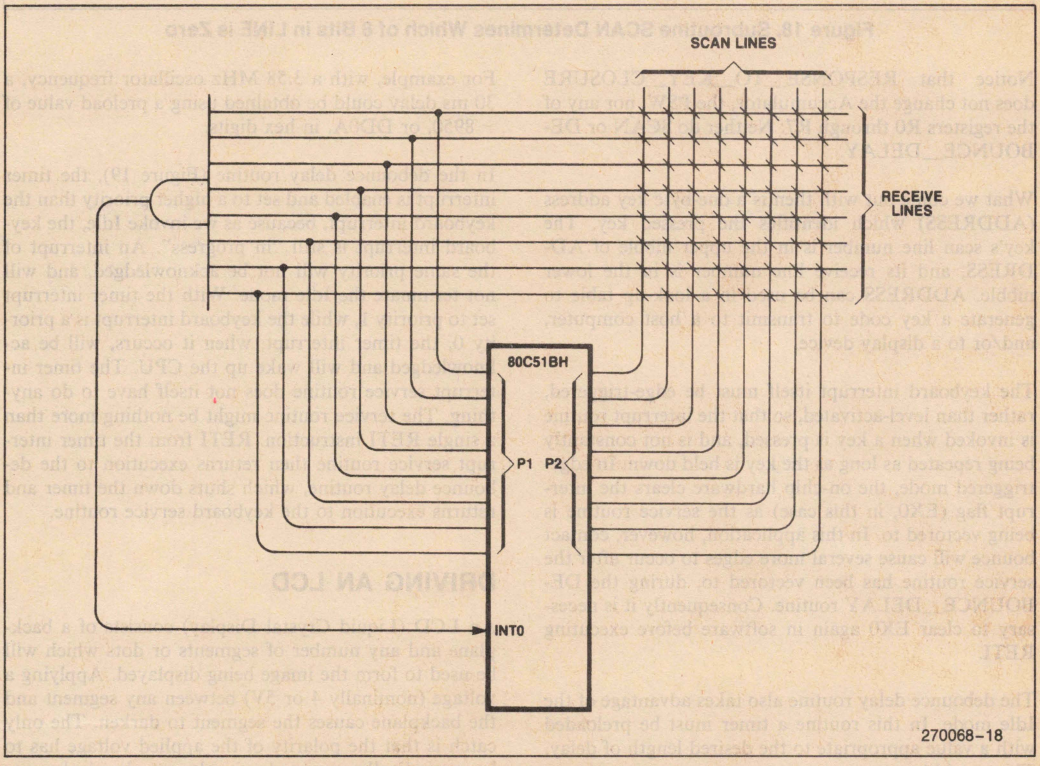


Figure 17. Scanning a Keyboard

```
SCAN:   MOV     ZERO_COUNTER,#0 ; ZERO_COUNTER counts the number of 0s in LINE.
        JB      LINE.0,ONE      ; Test LINE bit 0.
        INC     ZERO_COUNTER    ; If LINE.0 = 0, increment ZERO_COUNTER
        MOV     ZERO_BIT,#1     ;    and record that line number 1 is active.
ONE:    JB      LINE.1,TWO      ; Procedure continues for other LINE bits.
        INC     ZERO_COUNTER
        MOV     ZERO_BIT,#2     ; Line number 2 is active.
TWO:    JB      LINE.2,THREE
        INC     ZERO_COUNTER
        MOV     ZERO_BIT,#3     ; Line number 3 is active.
THREE:  JB      LINE.3,FOUR
        INC     ZERO_COUNTER
        MOV     ZERO_BIT,#4     ; Line number 4 is active.
FOUR:   JB      LINE.4,FIVE
        INC     ZERO_COUNTER
        MOV     ZERO_BIT,#5     ; Line number 5 is active.
FIVE:   JB      LINE.5,SIX
        INC     ZERO_COUNTER
        MOV     ZERO_BIT,#6     ; Line number 6 is active.
SIX:    JB      LINE.6,SEVEN
        INC     ZERO_COUNTER
        MOV     ZERO_BIT,#7     ; Line number 7 is active.
SEVEN:  JB      LINE.7,EIGHT
        INC     ZERO_COUNTER
        MOV     ZERO_BIT,#8     ; Line number 8 is active.
EIGHT:  RET
```

270068-19

Figure 18. Subroutine SCAN Determines Which of 8 Bits in LINE is Zero

Notice that RESPONSE_TO_KEY_CLOSURE does not change the Accumulator, the PSW, nor any of the registers R0 through R7. Neither do SCAN or DEBOUNCE_DELAY.

What we come out with then is a one-byte key address (ADDRESS) which identifies the pressed key. The key's scan line number is in the upper nibble of ADDRESS, and its receive line number is in the lower nibble. ADDRESS can be used in a look-up table to generate a key code to transmit to a host computer, and/or to a display device.

The keyboard interrupt itself must be edge-triggered, rather than level-activated, so that the interrupt routine is invoked when a key is pressed, and is not constantly being repeated as long as the key is held down. In edge-triggered mode, the on-chip hardware clears the interrupt flag (EX0, in this case) as the service routine is being vectored to. In this application, however, contact bounce will cause several more edges to occur after the service routine has been vectored to, during the DEBOUNCE_DELAY routine. Consequently it is necessary to clear EX0 again in software before executing RETI.

The debounce delay routine also takes advantage of the Idle mode. In this routine a timer must be preloaded with a value appropriate to the desired length of delay. This would be

$$\text{timer preload} = \frac{(\text{osc kHz}) \times (\text{delay time ms})}{12}$$

For example, with a 3.58 MHz oscillator frequency, a 30 ms delay could be obtained using a preload value of −8950, or DD0A, in hex digits.

In the debounce delay routine (Figure 19), the timer interrupt is enabled and set to a higher priority than the keyboard interrupt, because as we invoke Idle, the keyboard interrupt is still "in progress". An interrupt of the same priority will not be acknowledged, and will not terminate the Idle mode. With the timer interrupt set to priority 1, while the keyboard interrupt is a priority 0, the timer interrupt, when it occurs, will be acknowledged and will wake up the CPU. The timer interrupt service routine does not itself have to do anything. The service routine might be nothing more than a single RETI instruction. RETI from the timer interrupt service routine then returns execution to the debounce delay routine, which shuts down the timer and returns execution to the keyboard service routine.

DRIVING AN LCD

An LCD (Liquid Crystal Display) consists of a backplane and any number of segments or dots which will be used to form the image being displayed. Applying a voltage (nominally 4 or 5V) between any segment and the backplane causes the segment to darken. The only catch is that the polarity of the applied voltage has to be periodically reversed, or else a chemical reac-

```
DEBOUNCE_DELAY:
        MOV     TL1,#TL1_PRELOAD ; Preload low byte.
        MOV     TH1,#TH1_PRELOAD ; Preload high byte.
        SETB    ET1              ; Enable Timer 1 interrupt.
        SETB    PT1              ; Set Timer 1 interrupt to high priority.
        SETB    TR1              ; Start timer running.
        ORL     PCON,#1          ; Invoke Idle mode.
;
; The next instruction will not be executed until the delay times out.
;
        CLR     TR1              ; Stop the timer.
        CLR     PT1              ; Back to priority 0 (if desired).
        CLR     ET1              ; Disable Timer 1 interrupt (if desired).
        RET                      ; Continue keyboard scan.
```

270068-20

Figure 19. Subroutine DEBOUNCE_DELAY Puts the 80C51BH into Idle During the Delay Time

tion takes place in the LCD which causes deterioration and eventual failure of the liquid crystal.

To prevent this happening, the backplane and all the segments are driven with an AC signal, which is derived from a rectangular voltage waveform. If a segment is to be "off" it is driven by the same waveform as the backplane. Thus it is always at backplane potential. If the segment is to be "on" it is driven with a waveform that is the inverse of the backplane waveform. Thus it has about 5V of periodically changing polarity between it and the backplane.

With a little software overhead, the 80C51BH can perform this task without the need for additional LCD drivers. The only drawback is that each LCD segment uses up one port pin, and the backplane uses one more. If more than, say, two 7-segment digits are being driven, there aren't many port pins left for other tasks. Nevertheless, assuming a given application leaves enough port pins available to support this task, the considerations for driving the LCD are as follows.

Suppose, for example, it is a 2-digit display with a decimal point. One port (TENS_DIGIT) connects to the 7 segments of the tens digit plus the backplane. Another port (ONES_DIGIT) connects to a decimal point plus the 7 segments of the ones digit.

One of the 80C51BH's timers is used to mark off half-periods of the drive voltage waveform. The LCD drive waveform should have a rep rate between 30 and 100 Hz, but it's not very critical. A half-period of 12 ms will set the rep rate to about 42 Hz. The preload/reload value to get 12 ms to rollover is the 2's complement negative of the oscillator frequency in kHz: if the oscillator frequency is 3.58 MHz, the reload value is −3580, or F204 in hex digits.

Now, the 80C51BH would normally be in Idle, to conserve power, during the time that the LCD and other tasks are not requiring servicing. When the timer rolls over it generates an interrupt, which brings the 80C51BH out of Idle. The service routine reloads the timer (for the next rollover), and inverts the logic levels of all the pins that are connected to the LCD. It might look like this:

```
LCD_DRIVE_INTERRUPT:
  MOV   TL1,#LOW( - XTAL_FREQ)
  MOV   TH1,#HIGH( - XTAL_FREQ)
  XRL   TENS_DIGIT,#0FFH
  XRL   ONES_DIGIT,#0FFH
  RETI
```

To update the display, one would use a look-up table to generate the characters. In the table, "on" segments are represented as 1s, and "off" segments as 0s. The backplane bit is represented as a 0. The quantity to be displayed is stored in RAM as a BCD value. The look-up table operates on the low nibble of the BCD value, and produces the bit pattern that is to be written to either the ones digit or the tens digit. Before the new patterns can be written to the LCD, the LCD drive interrupt has to be disabled. That is to prevent a polarity reversal from taking place between the times the two digits are written. An update subroutine is shown in Figure 20.

USING AN LCD DRIVER

As was noted, driving an LCD directly with an 80C51BH uses a lot of port pins. LCD drivers are available in CMOS to interface an 80C51BH to a 4-digit display using only 7 of the C51BH's I/O pins. Basically, the C51BH tells the LCD driver what digit is to be displayed (4 bits) and what position it is to be displayed in (2 bits), and toggles a Chip Select pin to tell the driver to latch this information. The LCD driver generates the display characters (hex digits), and takes care of the polarity reversals using its own RC oscillator to generate the timing.

Figure 25 shows an 80C51BH working with an ICM7211M to drive a 4-digit LCD, and the software that updates the display.

One could equally well send information to the LCD driver over the bus. In that case, one would set up the Accumulator with the digit select and data input bits, and execute a MOVX@ R0,A instruction. The LCD driver's chip select would be driven by the CPU's $\overline{WR}$ signal. This is a little easier in software than the direct bit manipulation shown in Figure 21. However, it uses more I/O pins, unless there is already some external memory involved. In that case, no extra pins are used up by adding the LCD driver to the bus.

RESONANT TRANSDUCERS

Analog transducers are often used to convert the value of a physical property, such as temperature, pressure, etc., to an analog voltage. These kinds of transducers then require an analog-to-digital converter to put the measurement into a form that is compatible with a digital control system. Another kind of transducer is now becoming available that encodes the value of the physical property into a signal that can be directly read by a digital control system. These devices are called resonant transducers.

Resonant transducers are oscillators whose frequency depends in a known way on the physical property being measured. These devices output a train of rectangular pulses whose repetition rate encodes the value of the quantity being measured. The pulses can in most cases be fed directly into the 80C51BH, which then measures either the frequency or period of the incoming signal, basing the measurement on the accuracy of its own clock oscillator. The 80C51BH can even do this in its sleep; that is, in Idle.

When the frequency or period measurement is completed, the C51BH wakes itself up for a very short time to perform a sanity check on the measurement and convert it in software to any scaling of the measured quantity that may be desired. The software conversion can include corrections for nonlinearities in the transducer's transfer function.

Resolution is also controlled by software, and can even be dynamically varied to meet changing needs as a situation becomes more critical. For example, in a process controller you can increase your resolution ("fine tune" the control, as it were) as the process approaches its target.

The nominal reference frequency of the output signal from these devices is in the range of 20 Hz to 500 kHz, depending on the design. Transducers are available that have a full scale frequency shift 2 to 1. The transducer operates from a supply voltage range of 3V to 20V, which means it can operate from the same supply voltage as the 80C51BH. At 5V, the transducer draws less than 5 mA (Reference 7). It can normally be connected directly to one of the C51BH's port pins, as shown in Figure 22.

FREQUENCY MEASUREMENTS

Measuring a frequency means counting pulses for a known sample time. Two timer/counters can be used, one to mark off the sample time and one to count pulses. If the frequency being counted doesn't exceed 50 kHz or so, one may equally well connect the transducer signal to one of the external interrupt pins, and count pulses in software. That frees up one timer, with very little cost in CPU time.

The count that is directly obtained is TxF, where T is the sample time and F is the frequency. The full scale

```
UPDATE_LCD:
        CLR     ET1                     ; Disable LCD drive interrupt.
        MOV     DPTR,#TABLE_ADDRESS     ; Look-up table begins at TABLE_ADDRESS
        MOV     A,BCD_VALUE             ; Digits to be displayed.
        SWAP    A                       ; Move tens digit to low nibble.
        ANL     A,#0FH                  ; Mask off high nibble.
        MOVC    A,@A+DPTR               ; Tens digit pattern to accumulator.
        MOV     TENS_DIGIT,A            ; Update LCD tens digit.
        MOV     A,BCD_VALUE             ; Digits to be displayed.
        ANL     A,#0FH                  ; Mask off tens digit.
        MOVC    A,@A+DPTR               ; Ones digit pattern to accumulator.
        MOV     C,DECIMAL_POINT         ; Add decimal point to segment
        MOV     ACC.7,C                 ;  pattern.  Update LCD decimal point
        MOV     ONES_DIGIT,A            ;  and ones digit.
        SETB    ET1                     ; Re-enable LCD drive interrupt.
        RET
```

270068-21

Figure 20. UPDATE_LCD Routine Writes Two Digits to an LCD

range is Tx(Fmax-Fmin). For n-bit resolution

$$1\ LSB = \frac{Tx(Fmax-Fmin)}{2^n}$$

Therefore the sample time required for n-bit resolution is

$$T = \frac{2^n}{Fmax-Fmin}$$

For example, 8-bit resolution in the measurement of a frequency that varies between 7 kHz and 9 kHz would require, according to this formula, a sample time of 128 ms. The maximum acceptable frequency count would be 128 ms × 9 kHz = 1152 counts. The minimum would be 896 counts. Subtracting 896 from each frequency count (or presetting the frequency counter to −896 = 0FC80H) would allow the frequency to be reported on a scale of 0 to FF in hex digits.

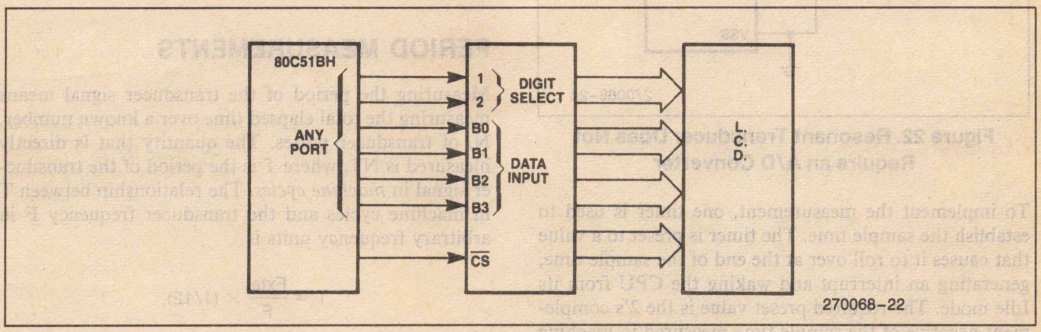


Figure 21a. Using an LCD Driver

```
UPDATE_LCD:
        MOV     A,DISPLAY_HI    ; High byte of 4-digit display.
        SETB    DIGIT_SELECT_2  ; Select leftmost digit of LCD.
        SETB    DIGIT_SELECT_1  ;  (Digit address = 11B.)
        CALL    SHIFT_AND_LOAD  ; High nibble of high byte to selected digit
        CLR     DIGIT_SELECT_1  ; Select second digit of LCD (address = 10B)
        CALL    SHIFT_AND_LOAD  ; Low nibble of high byte to selected digit.
        MOV     A,DISPLAY_LO    ; Low byte of 4-digit display.
        CLR     DIGIT_SELECT_2  ; Select third digit of LCD.
        SETB    DIGIT_SELECT_1  ;   (Digit address = 01B.)
        CALL    SHIFT_AND_LOAD  ; High nibble of low byte to selected digit.
        CLR     DIGIT_SELECT_1  ; Select fourth digit (address = 00B).
        CALL    SHIFT_AND_LOAD  ; Low nibble of low byte to selected digit.
        RET

SHIFT_AND_LOAD:
        RLC     A                       ; MSB to carry bit (CY).
        MOV     DATA_INPUT_B3,C         ; CY to Data Input pin B3.
        RLC     A                       ; Next bit to CY.
        MOV     DATA_INPUT_B2,C         ; CY to Data Input pin B2.
        RLC     A                       ; Next bit to CY.
        MOV     DATA_INPUT_B1,C         ; CY to Data Input pin B1.
        RLC     A                       ; Last bit to CY.
        MOV     DATA_INPUT_B0,C         ; CY to Data Input pin B0.
        CLR     CHIP_SELECT             ; Toggle Chip Select.
        SETB    CHIP_SELECT             ; 0-to-1 transition latches info.
        RET
```

270068-23

Figure 21b. UPDATE_LCD Routine Writes 4 Digits to an LCD Driver

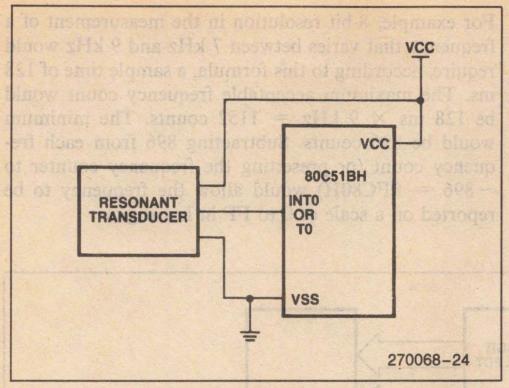


Figure 22. Resonant Transducer Does Not Require an A/D Converter

To implement the measurement, one timer is used to establish the sample time. The timer is preset to a value that causes it to roll over at the end of the sample time, generating an interrupt and waking the CPU from its Idle mode. The required preset value is the 2's complement negative of the sample time measured in machine cycles. The conversion from sample time to machine cycles is to multiply it by 1/12 the clock frequency. For example, if the clock frequency is 12 MHz, then a sample time of 128 ms is

(128 ms) × (12000 kHz)/12 = 128000 machine cycles.

Then the required preset value to cause the timer to roll over in 128 ms is

−128000 = FE0C00, in hex digits.

Note that the preset value is 3 bytes wide whereas the timer is only 2 bytes wide. This means the timer must be augmented in software in the timer interrupt routine to three bytes. The 80C51BH has a DJNZ instruction (decrement and jump if not zero) that makes it easier to code the third timer byte to count down instead of up. If the third timer byte counts down, its reload value is the 2's complement of what it would be for an up-counter. For example, if the 2's complement of the sample time is FE0C00, then the reload value for the third timer byte would be 02, instead of FE. The timer interrupt routine might then be:

```
TIMER_INTERRUPT_ROUTINE:
        DNJZ    THIRD_TIMER_BYTE,OUT
        MOV     TL0,#0
        MOV     TH0,#0CH
        MOV     THIRD_TIMERBYTE,#2
        MOV     FREQUENCY,COUNTER_L0
;Preset COUNTER to -896:
        MOV     COUNTER_L0,#80H
        MOV     COUNTER_HI,#0FCH
OUT:    RETI
```

At this point the value of the frequency of the transducer signal, measured to 8 bit resolution, is contained in FREQUENCY. Note that the timer can be reloaded on the fly. Note too that the timer can be reloaded on the fly. Note too that for 8-bit resolution only the low byte of the frequency counter needs to be read, since the high byte is necessarily 0. However, one may want to test the high byte to ensure that it is zero, as a sanity check on the data. Both bytes, of course must be reloaded.

PERIOD MEASUREMENTS

Measuring the period of the transducer signal means measuring the total elapsed time over a known number, N, of transducer pulses. The quantity that is directly measured is NT, where T is the period of the transducer signal in *machine cycles*. The relationship between T in machine cycles and the transducer frequency F in arbitrary frequency units is

$$T = \frac{Fxtal}{F} \times (1/12),$$

where Fxtal is the 80C51BH clock frequency, in the same units as F.

The full scale range then is Nx (Tmax−Tmin). For n-bit resolution.

$$1\ LSB = \frac{Ns(Tmax\text{-}Tmin)}{2^n}.$$

Therefore the number of periods over which the elapsed time should be measured is

$$N = \frac{2^n}{Tmax\text{-}Tmin}$$

However, N must also be an integer. It is logical to evaluate the above formula (don't forget Tmax and Tmin have to be in machine cycles) and select for N the next higher integer. This selection gives a period measurement that has somewhat more than n-bit resolution, but it can be scaled back if desired.

For example, suppose we want 8-bit resolution in the measurement of the period of a signal whose frequency varies from 7.1 kHz to 9 kHz. If the clock frequency is 12 MHz, then Tmax is (12000 kHz/7.1 kHz) x (1/12) = 141 machine cycles. Tmin is 111 machine cycles. The required value for N, then, is 256/(141-111) = 8.53 periods, according to the formula. Using N = 9 periods will give a maximum NT value of 141 x 9 = 1269 machine cycles. The minimum NT will be 111 × 9 = 999 machine cycles. A lookup table can be used to

scale these values back to a range of 0 to 255, giving precisely the 8-bit resolution desired.

To implement the measurement, one timer is used to measure the elapsed time, NT. The transducer is connected to one of the external interrupt pins, and this interrupt is configured to the transition-activated mode. In the transition-activated mode every 1-to-0 transition in the transducer output will generate an interrupt. The interrupt routine counts transducer pulses, and when it gets to the predetermined N, it reads and clears the timer. For the specific example cited above, the interrupt routine might be:

```
INTERRUPT_RESPONSE:
          DJNZ    N,OUT
          MOV     N,#9
          CLR     EA
          CLR     TR1
          MOV     NT_LO,TL1
          MOV     NT_HI,TH1
          MOV     TL1,#9
          MOV     TH1,#0
          SETB    TR1
          SETB    EA
          CALL    LOOKUP_TABLE
OUT:      RETI
```

In this routine a pulse counter N is decremented from its preset value, 9, to zero. When the counter gets to zero it is reloaded to 9. Then all interrupts are blocked for a short time while the timer is read and cleared. The timer is stopped during the read and clear operations, so "clearing" it actually means presetting it to 9, to make up for the 9 machine cycles that are missed while the timer is stopped.

The subroutine LOOKUP_TABLE is used to scale the measurement back to the desired 8-bit resolution. It can also include built-in corrections for errors or nonlinearities in the transducer's transfer function.

The subroutine uses the MOVC A, @ A + DPTR instruction to access the table, which contains 270 entries commencing at the 16-bit address referred to as TABLE. The subroutine must compute the address of the table entry that corresponds to the measured value of NT. This address is

$$DPTR = TABL + NT - NTMIN,$$

where NTMIN = 999, in this specific example.

```
LOOKUP_TABLE:
          PUSH  ACC
          PUSH  PSW
          MOV   A,#LOW(TABLE-NTMIN)
          ADD   A,NT_LO
          MOV   DPL,A
          MOV   A,#HIGH(TABLE-NMTIN)
          ADDC  A,NT_HI
          MOV   DPH,A
          CLR   A
          MOVC  A,@A+DTPR
          MOV   PERIOD,A
          POP   PSW
          POP   ACC
          RET
```

At this point the value of the period of the transducer signal, measured to 8 bit resolution, is contained in PERIOD.

PULSE WIDTH MEASUREMENTS

The 80C51BH timers have an operating mode which is particularly suited to pulse width measurements, and will be useful in these applications if the transducer signal has a fixed duty cycle.

In this mode the timer is turned on by the on-chip circuitry in response to an input high at the external interrupt pin, and off by an input low, and it can do this while the 80C51BH is in Idle. (The "GATE" mode of timer operation is described in the Intel Microcontroller Handbook.) The external interrupt itself can be enabled, so the same 1-to-0 transition from the transducer that turns off the timer also generates an interrupt. The interrupt routine then reads and resets the timer.

The advantage of this method is that the transducer signal has direct access to the timer gate, with the result that variations in interrupt response time have no effect on the measurement.

Resonant transducers that are designed to fully exploit the GATE mode have an internal divide-by-N circuit that fixes the duty cycle at 50% and lowers the output frequency to the range of 250 to 500 Hz (to control RFI). The transfer function between transducer period and measurand is approximately linear, with known and repeatable error functions.

HMOS/CHMOS Interchangeability

The CHMOS version of the 8051 is architecturally identical with the HMOS version, but there are nevertheless some important differences between them which the designer should be aware of. In addition, some applications require interchangeability between HMOS and CHMOS parts. The differences that need to be considered are as follows:

External Clock Drive: To drive the HMOS 8051 with an external clock signal, one normally grounds the XTAL1 pin and drives the XTAL2 pin. To drive the CHMOS 8051 with an external clock signal, one must drive the XTAL1 pin and leave the XTAL2 pin unconnected. The reason for the difference is that in the

HMOS 8051, it is the XTAL2 pin that drives the internal clocking circuits, whereas in the CHMOS version it is the XTAL1 pin that drives the internal clocking circuits.

There are several ways to design an external clock drive to work with both types. For low clock frequencies (below 6 MHz), the HMOS 8051 can be driven in the same way as the CHMOS version, namely, through XTAL1 with XTAL2 unconnected. Another way is to drive both XTAL1 **and** XTAL2; that is, drive XTAL1 and use and external inverter to derive from XTAL1 a signal with which to drive XTAL2.

In either case, a 74HC or 74HCT circuit makes an excellent driver for XTAL1 and/or XTAL2, because neither the HMOS nor the CHMOS XTAL pins have TTL-like input logic levels.

Unused Pins: Unused pins of Ports 1, 2 and 3 can be ignored in both HMOS and CHMOS designs. The internal pullups will put them into a defined state. Unused Port 0 pins in 8051 applications can be ignored, even if they're floating. But in 80C51BH applications, these pins should not be left afloat. They can be externally pulled up or down, or they can be internally pulled down by writing 0s to them.

8031/80C31BH designs may or may not need pullups on Port 0. Pullups aren't needed for program fetches, because in bus operations the pins are actively pulled high or low by either the 8031 or the external program memory. But they are needed for the CHMOS part if the Idle or Power Down mode is invoked, because in these modes Port 0 floats.

Logic Levels: If V_{CC} is between 4.5V and 5.5V, an input signal that meets the HMOS 8051's input logic levels will also meet the CHMOS 80C51BH's input logic levels (except for XTAL1/XTAL2 and RST). For the same V_{CC} condition, the CHMOS device will reach or surpass the output logic levels of the HMOS device. The HMOS device will not necessarily reach the output logic levels of the CHMOS device. This is an important consideration if HMOS/CHMOS interchangeability must be maintained in an otherwise CMOS system.

HMOS 8051 outputs that have internal pullups (Ports 1, 2, and 3) "typically" reach 4V or more if I_{OH} is zero, but not fast enough to meet timing specs. Adding an external pullup resistor will ensure the logic level, but still not the timing, as shown in Figure 23. If timing is an issue, the best way to interface HMOS to CMOS is through a 74HCT circuit.

Idle and Power Down: The Idle and Power Down modes exist only on the CHMOS devices, but if one

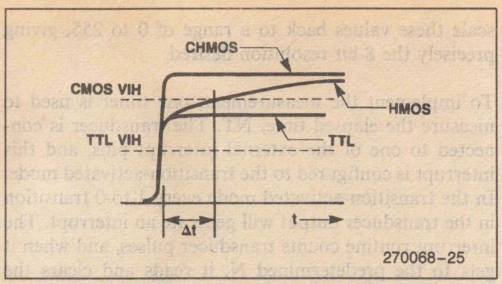


Figure 23. 0-to-1 Transition Shows Unspec'd Delay (Δt) in HMOS to 74HC Logic

wishes to preserve the capability of interchanging HMOS and CHMOS 8051s the software has to be designed so that the HMOS parts will respond in an acceptable manner when a CHMOS reduced power mode is invoked.

For example, an instruction that invokes Power Down can be followed by a "JMP $":

```
CLR   EA
ORL   PCON,#2
JMP   $
```

The CHMOS and HMOS parts will respond to this sequence of code differently. The CHMOS part, going into a normal CHMOS Power Down Mode, will stop fetching instructions until it gets a hardware reset. The HMOS part will go through the motions of executing the ORL instruction, and then fetch the JMP instruction. It will continue fetching and executing JMP $ until hardware reset.

Maintaining HMOS/CHMOS 8051 interchangeability in response to Idle requires more planning. The HMOS part will not respond to the instruction that puts the CHMOS part into Idle, so that instruction needs to be followed by a software idle. This would be an idling loop which would be terminated by the same conditions that would terminate the CHMOS's hardware Idle. Then when the CHMOS device goes into Idle, the HMOS version executes the idling loop, until either a hardware reset or an enabled interrupt is received. Now if Idle is terminated by an interrupt, execution for the CHMOS device will proceed after RETI from the instruction following the one that invoked Idle. The instruction following the one that invoked Idle is the idling loop that was inserted for the HMOS device. At this point, both the HMOS and CHMOS devices must be able to fall through the loop to continue execution.

One way to achieve the desired effect is to define a "fake" Idle flag, and set it just before going into Idle. The instruction that invoked Idle is followed by a software idle:

```
SETB    IDLE
ORL     PCON,#1
JB      IDLE,$
```

Now the interrupt that terminates the CHMOS's Idle must also break the software idle. It does so by clearing the "Idle" bit:

```
...
CLR   IDLE
RETI
```

Note too that the PCON register in the HMOS 8051 contains only one bit, SMOD, whereas the PCON register in CHMOS contains SMOD plus four other bits. Two of those other bits are general purpose flags. Maintaining HMOS/CHMOS interchangeability requires that these flags not be used.

REFERENCES

1. Pawlowski, Moroyan, Alnether, "Inside CMOS Technology," *BYTE magazine,* Sept., 1983. Available as Article Reprint AR-302.
2. Kokkonen, Pashley, "Modular Approach to C-MOS Technology Tailors Process to Application," *Electronics,* May, 1984. Available as Article Reprint AR-332.
3. Williamson, T., *Designing Microcontroller Systems for Electrically Noisy Environments,* Intel Application Note AP-125, Feb. 1982.
4. Williamson, T., "PC Layout Techniques for Minimizing Noise," *Mini-Micro Southeast,* Session 9, Jan., 1984.
5. Alnether, J., *High Speed Memory System Design Using 2147H, Intel Application Note AP-74, March 1980.*
6. Ott, H., "Digital Circuit Grounding and Interconnection," *Proceedings of the IEEE Symposium on Electromagnetic Compatibility,* pp. 292–297, Aug. 1981.
7. *Digital Sensors by Technar,* Technar Inc., 205 North 2nd Ave., Arcadia, CA 91006.

One way to achieve the desired effect is to define a "take" Idle flag, and set it just before going into Idle. The instruction that invoked Idle is followed by a software Idle:

```
SETB    IDLE
ORL     PCON,#1
JB      IDLE,$
```

Now the interrupt that terminates the CHMOS's Idle must also break the software Idle. It does so by clearing the "Idle" bit.

```
...
CLR     IDLE
RETI
```

Note too that the PCON register in the HMOS 8051 contains only one bit, SMOD, whereas the PCON register in CHMOS contains SMOD plus four other bits. Two of these other bits are general purpose flags. Maintaining HMOS/CHMOS interchangeability requires that these flags not be used.

REFERENCES

1. Pawlowski, Morgan, Altmiller, "Inside CMOS Technology," *BYTE* magazine Sept., 1983. Available as Article Reprint AR-302.

2. Kotkoski, Bashley, "Modular Approach to C-MOS Technology Tailors Process to Application," *Electronics*, May, 1984. Available as Article Reprint AR-323.

3. Williamson, T., Designing Microcontroller Systems for Electrically Noisy Environments, Intel Application Note AP-125, Feb. 1982.

4. Williamson, T., "PC Layout Techniques for Minimizing Noise," *Mini-Micro Southeast*, Session 9, Jan., 1984.

5. Altmiller, J., High Speed Memory System Design Using 27C64, Intel Application Note AP-74, March 1982.

6. Orr, B., "Digital Circuit Grounding and Interconnection," *Proceedings of the IEEE Symposium on Electromagnetic Compatibility*, pp. 272–277, Aug. 1981.

7. Digital Scopes by Tektronix Technical Inc., 305 North 2nd Ave., Arcadia, CA 91006.

MCS®-51 Article Reprints

10

intel

ARTICLE REPRINT

AR-374

January 1985

Built-In Basic Interpreter Turns Controller Chip Into Versatile System Core

John Katausky
Intel Corporation
Chandler, AZ

ORDER NUMBER: 270051-001
270051-1

DESIGN ENTRY

Built-in Basic interpreter turns controller chip into versatile system core

An extended set of Basic statements and operators transforms a chip into the foundation for a wide range of embedded real-time systems.

Single-chip computers have been popular for some time now, especially in embedded applications. But the phrase "computer on a chip" takes on a whole new meaning with the appearance of a software package that implements a version of Basic in silicon.

The 8052AH-Basic controller is aimed primarily at data acquisition, test instrumentation, and process control. Further, it is right at home in virtually any embedded environment and monitoring system.

Unlike so-called Tiny Basic interpreters, the chip's MCS Basic-52 software package is a full Basic interpreter. In addition, it can manipulate strings and handle logical operators (AND, OR, exclusive-OR, and NOT) and floating-point arithmetic. It is also able to accept and deliver numbers in floating-point, integer, or hexadecimal formats. Counted among its other features are built-in EPROM programming, mnemonic access to all on-chip I/O resources, and a built-in real-time clock with a resolution to within 5 ms. Moreover, its complete function library of routines can be accessed in assembly language (see the table, opposite). And the package supplements Basic's standard instructions with commands designed specifically for embedded systems. The package resides in the 8-kbit ROM of the 8052AH-Basic chip. Apart from the software, it is identical with the 8052AH and so features 256 bytes of RAM and three multiple-mode 16-bit timer-counters.

Although Basic is held in low regard by many computer scientists, one fact cannot be denied: It is by far the most popular microcomputer language. Not only is virtually every personal or home computer capable of running it, but the language has become a de facto standard for many applications, including laboratory and

John Katausky, Intel Corp.

John Katausky is technical marketing manager for Intel Corp.'s microcontroller operation in Chandler, Ariz.

Summary of Basic-52's features

Commands	Statements		Operators	
RUN	BAUD	INPUT	ADD (+)	ASC ( )
LIST	CALL	LET	DIVIDE (/)	CHR ( )
LIST#	CLEAR	ONERR	EXPONENTIATION (**)	CBY ( )
NEW	**CLEARS**	**ONEXT1**	MULTIPLY (*)	DBY ( )
NULL	**CLEAR1**	**ONTIME**	SUBTRACT (−)	XBY ( )
RAM	**CLOCK0**	PRINT	LOGICAL AND (.AND.)	GET
ROM	**CLOCK1**	**PRINT#**	LOGICAL OR (.OR.)	**IE**
XFER	DATA	**PH0.**	LOGICAL X-OR (.XOR.)	**IP**
PROG	READ	**PH0.#**	LOGICAL NOT	**PORT1**
PROG1	RESTORE	**PH1.**	ABS ( )	**PCON**
PROG2	DIM	**PH1.#**	INT ( )	**RCAP2**
FPROG	**DO-WHILE**	**PUSH**	SGN ( )	**T2CON**
FPROG1	**DO-UNTIL**	**POP**	SQR ( )	**TCON**
FPROG2	END	**PWM**	RND	**TMOD**
	FOR-TO-STEP	REM	LOG ( )	**TIME**
	NEXT	**RETI**	EXP ( )	**TIMER0**
	GOSUB	STOP	SIN ( )	**TIMER1**
	RETURN	**STRING**	COS ( )	**TIMER2**
	GOTO	**UI0**	TAN ( )	**TIME**
	ON-GOTO	**UI1**	ATN ( )	**XTAL**
	ON-GOSUB	**UO0**	=,>,>=,	**MTOP**
		UO1	<,<=,<>	LEN
	IF-THEN-ELSE		**PI**	FREE

Boldface statements have been added specifically for embedded systems.

Reprinted from **ELECTRONIC DESIGN - December 13, 1984**

Copyright 1984 Hayden Publishing Co., Inc.

office automation. Its popularity is easily explained by its simplicity and ease of use. A programmer can execute, edit, and debug short programs in minutes, without having to worry about compilation, linking, and locating.

Building up Basic

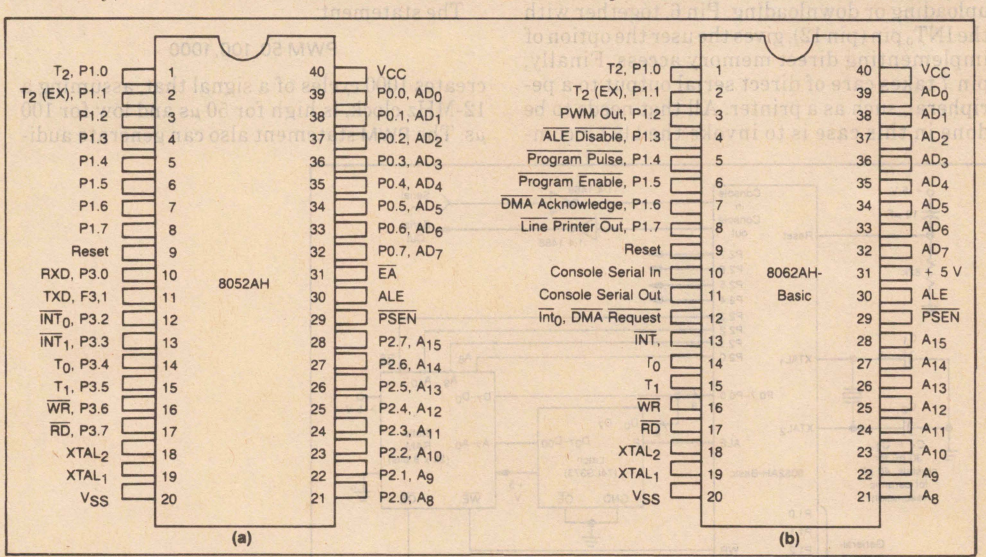
Like any language, Basic has its drawbacks, but the 8052AH-Basic's software package addresses many of them by beefing up its instruction set. These enhancements address three general areas: First, sophisticated programmers dislike the fact that Basic is an unstructured language. Because GO TO statements can be placed anywhere in a program, understanding and debugging it can become quite a chore. For that reason, the package contains the more structured DO-WHILE and DO-UNTIL statements, as well as the standard FOR-NEXT and IF-THEN-ELSE. These extensions furnish better ways to control the flow of a program.

Second, the fact that Basic uses global variables forces subroutines to use the same names for variables as the main program employs. Generally, this is an inconvenience because variables may need to be redefined every time a subroutine is called up. The package's function library, however, includes PUSH and POP statements that allow the user to pass variables to subroutines on a stack, making it easy to redefine variables in a subroutine. Additionally, PUSH and POP can be used to pass parameters to and from user-supplied assembly language routines.

Finally, interpreted Basic is usually a slow language. Although this is a legitimate complaint, many applications can still be adequately served by it. The package's fast token-based interpreter, though, puts an end to the problem, turning in execution speeds that compare quite favorably with those of popular 8- and 16-bit personal computers.

A quick turn around the pins

As mentioned, the software is stored in an 8052AH controller chip, which is itself housed in a standard 40-pin DIP (Fig. 1). Some of the pins' uses are predetermined. Of the four ports on the standard 8052AH, only port 1 still serves as an I/O port on the Basic version. Since the



1. The pin assignments of the 8052AH (a) and the 8052AH-Basic controller (b) differ primarily in that the second does not use lines AD_0 to AD_7 and A_8 to A_{15} as ports 0 and 2, respectively. Further, the Basic version needs +5 V on pin 31 to enable the 8052AH to execute from internal ROM.

270051-3

8052AH-Basic requires external memory, Pins 32 through 39 and 21 through 28, which formerly served ports 0 and 2 respectively, are now used for their alternative function: addressing and exchanging data with RAM and ROM. Pins 10 through 17, which formed port 3, are putting in their time at their assorted alternative duties—with some slight revisions. The only other changes involve pin 31, $\overline{EA}$, which is now tied to V_{CC}, and port 1, whose pins have acquired a second set of functions.

Port 1 in detail

To start with, pins 1 and 2 of port 1 (P1.0 and P1.1) can also be used to clock and trigger timer-counter 2. Pins 3, 4, and 5 generate all timing and other signals necessary to program just about any EPROM or EEPROM—a simple way to save and retrieve programs without uploading or downloading. Pin 6, together with the INT_0 pin (pin 12), gives the user the option of implementing direct memory access. Finally, pin 7 takes care of direct serial output to a peripheral, such as a printer. All that needs to be done in this case is to invoke the LIST# command or the PRINT# statement.

In addition, the syntax of Basic-52 permits the user to read and write directly from and to port 1. For instance, the statement:

```
PORT1 = 55H
```

would place the hexadecimal value 55 on the appropriate pins, while:

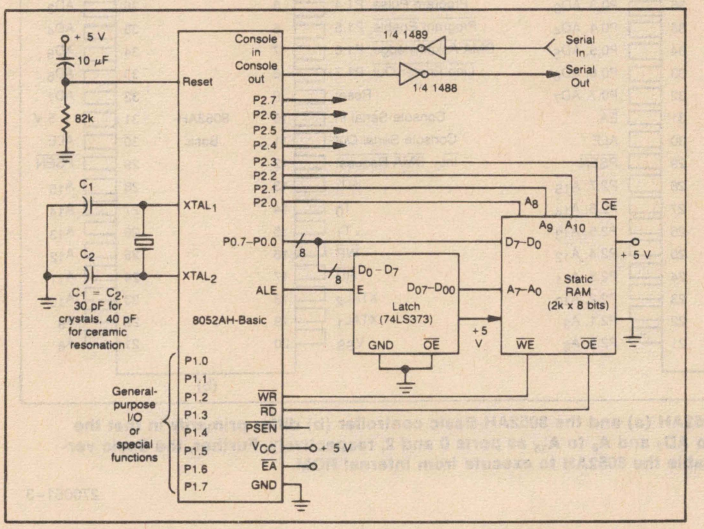
```
A = PORT1
```

would assign the value of variable A to the port 1 pins. Furthermore, some of the language's commands and statements manipulate those pins so that they furnish a number of useful and unique features. For example, P1.2 can also serve as an output for the pulse-width modulate (PWM) statement. Executing it generates a pulse train of varying duration, frequency, and duty cycle on the pin.

The statement:

```
PWM 50,100,1000
```

creates 1000 cycles of a signal that, assuming a 12-MHz clock, is high for 50 μs and low for 100 μs. The PWM statement also can generate audi-



2. A simple system based on the 8052AH-Basic microprocessor requires just three additional chips plus the serial-port driver. Port 1 could be used, for example, to sense switches, drive indicators, or a power printer.

270051-4

ble feedback in process control and security systems.

To pin 9 (Reset) falls the job of initializing the system. It also furnishes power on reset when an external 8.2-kΩ resistor is connected from this pin to ground and a 10-μF capacitor is connected to V_{CC}.

Under the software package, all of the pins that formed port 3 are pressed into service in their alternative functions. One of the interrupt pins, INT_0, now also handles DMA requests (pin 12), and the pins for transmitting and receiving data (pins 11 and 12) now handle serial inputs and outputs to the console. Their activities are integrated into a Basic application program. For example, an interrupt on the INT_1 pin (pin 13) can be handled by the ONEXT1 statement:

```
10 ONEXT1 1000
```

which forces a GOSUB to line 1000 every time the INT_1 pin is pulled to a logical 0. In the called subroutine, the user can process the interrupt, then use a RETI statement to exit from the interrupt handler.

To turn the Basic chip into a complete system, the user need supply only external RAM and serial port drivers (Fig. 2). The RAM requirements are few: At least 1 kbyte must be present, and it must start at external memory location 0 and be contiguous and completely decoded. After reset, the package determines how much memory is present in the system and initializes it, then waits for the user to type in a space character. That character establishes the baud rate automatically.

Although this system is quite useful, it does not represent an ideal embedded controller. Fortunately, adding just a little more hardware is all that is needed to construct one complete with automatic self-starting, EPROM and EEPROM programming, and simple program-file management (see "EPROM Saves the Program," opposite).

Once the desired type of EPROM is in place, starting up the system involves no more than entering either the PROG2 or FPROG2 command. Both commands preserve the baud rate information in exactly the same way as PROG1 or FPROG1 does. In addition, though, either causes the first program stored in PROM to ex-

EPROM saves the program

EPROM and EEPROM programming is certainly one of the most powerful features of the MCS Basic-52 software package. And it is simple to use; only three of the 8052AH-Basic microprocessor's control pins are needed to call it into play. In brief, after the user enters a program, it is stored in RAM, starting at location 512. When the user types PROG PROG or FPROG, the program that is stored in RAM is programmed into an EPROM or EEPROM whose address is 8000_{16}.

To start the process, the CPU first writes a logical 0 to the Program Enable pin, P1.5, thus supplying the higher voltage required to program the EPROM. Next, the processor reads the appropriate RAM location, starting at 512, and saves the byte of information.

Then, the CPU sends out the appropriate low-order PROM address to pins AD_0 to AD_7 and sets the ALE Disable pin to 0. That latches the low-order PROM address permanently into the system address latch—74LS373 (see the figure). Only one AND gate is required to do this. The processor then writes the high-order PROM address to pins A_8 through A_{15} and the data to pins AD_0 to AD_7. Finally, a 0 is written to the Program Pulse pin. Depending on the PROM used, that pin is held low for either 1 or 50 ms. After it returns to a logical 1, the CPU verifies the contents of the programmed PROM and then writes a 1 to ALE Disable.

The process continues until the entire Basic program is saved in PROM. When programming is complete, the CPU writes a 1 to the Progam Enable pin, leaves the programming routine, and returns to the Basic command mode. To prevent accidental programming during power-up, OR gates should be used (top left in figure).

The foregoing description of the programming naturally leads to two questions.

The first concerns the duration of the 0 state. A 50-ms pulse is required to program most standard EPROMs. The 1-ms pulse, on the other hand, permits the software package to program Intel's latest generation of high-density EPROMs, working with the company's programming algorithm, dubbed INTELigent. To apply it, the user simply types FPROG instead of PROG. The algorithm requires that the V_{CC} on the EPROM be increased to 6 V. One economical way to accomplish this is with a DIP relay. The software package also allows the user to specify programming pulses of any length.

The other question centers on the accuracy of the programming pulse versus that of the system clock frequency. To eliminate any dependency on the

270051-5

DESIGN ENTRY

Basic controller

system clock, the package actually calculates all critical timing parameters from a user-supplied variable, XTAL. After a reset, the package assumes a system clock frequency of 11.059200 MHz unless told otherwise. If the user enters the statement:

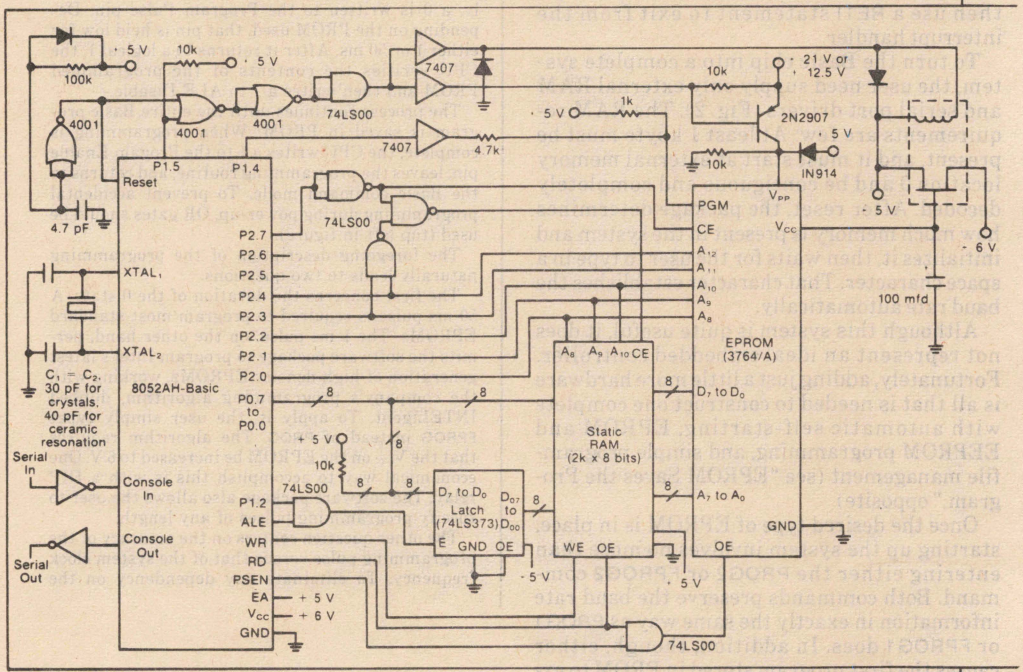
XTAL = 12000000

a 12-MHz system clock is assumed and the proper PROM programming time calculated. The real-time clock pulses and the baud rate for the line printer output port are also determined. This ability gives the designer the freedom to choose the desired system clock frequency without having to worry about the internal timing of the Basic software.

Another useful aspect of the software's PROM programming capabilities is that it stores up to 255 programs in an EPROM. PROM size, of course, limits the possible number of programs. Every time PROG (or FPROG) is typed, the software responds with the file number that has been assigned to that program. To call up a given program, the user simply types ROM X, and the package finds the appropriate one. Typing RUN starts program execution.

Suppose the user discovers an error in a program that is stored in EPROM—normally bad news because such programs cannot be edited. However, the package allows a program to be transferred from EPROM to RAM by typing XFER. Once XFER is entered, the program may be edited like any other. Typing PROG again saves the edited program.

Additionally, after reset, the software package waits for a space character, from which the baud rate is derived, to be entered on the serial port. As an option, the package can be made, on receiving the command PROG1 or FPROG1, to save the serial-port baud rate information in PROM. The next time the processor is reset, the package can sign on directly without the need to enter a space character.



270051-6

ecute directly after reset, bypassing a RUN command. In fact, even the console is no longer needed in some embedded systems. Programs can be written and debugged with the aid of a terminal, and when the programmer is satisfied with the results, the hardware can be embedded in the design and the terminal disconnected.

On call

Aside from the features already noted, the package contains a host of others that are very attractive for embedded systems. The CALL statement, for instance, accesses assembly-language routines directly from the package, a handy way in which to meet speed requirements. Better still, assembly-language programs can take advantage of a complete library of routines that reside in the package. The user gains access to this library simply by placing a specific op code in the accumulator and then sending a CALL to location 30_{16}.

Suppose, for example, that the assembly-language routine reads a 16-bit value, and the user would like to calculate the value's square root and send it to a particular location. Assuming that the user supplies the 16-bit integer to the high- and low-byte registers, R_2 and R_0, respectively, the calculation can be made simply (Program 1). All told, over 60 routines can be invoked by assembly-language programs.

The software package also controls all of the system's I/O memory resources. To make things simple, all of these operations are symmetrical. For example, the operator DBY in the statement:

```
A = DBY (0FEH)
```

assigns the value located at OFE_{16} in the chip's internal memory to variable A. The statement:

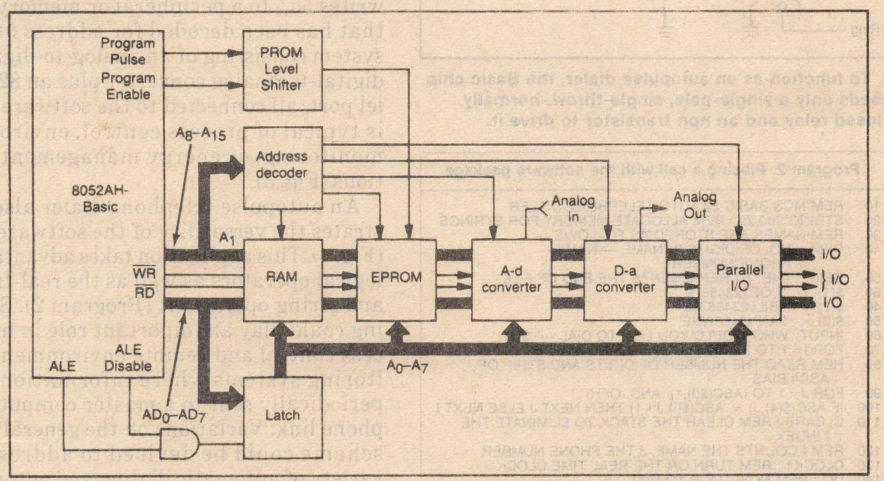
```
DBY (0FEH) = 22H
```

places 22_{16} in internal memory location $0FE_{16}$. The same symmetry holds for the interrupt registers, IE and IP, as well as the timers and the timer configuration registers.

Other special instructions are particularly

Program 1. Getting to the root of a variable

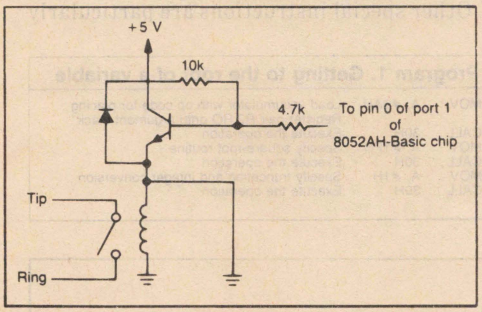
```
MOV     A, #9AH     ;Load accumulator with op code for placing
                    ;Register pair R2:R0 onto argument stack
CALL    30H         ;Execute the operation
MOV     A, #1FH     ;Specify square-root routine
CALL    30H         ;Execute the operation
MOV     A, #1H      ;Specify truncation and integer conversion
CALL    30H         ;Execute the operation
```



3. With the help of an address decoder, the I/O on a system built around the Basic chip can be expanded at will. One such setup converts analog into digital and digital into analog signals, and supplies a parallel I/O port for a console.

270051-7

helpful for the designer. Thus BAUD sets the baud rate for the line printer port. If the designer wants to transmit at 1200 bauds, he or she simply enters BAUD 1200. CLEARS is used to reset all of the package's stacks, and CLEARLI resets all interrupts, except the real-time clock. CLOCK1 enables the real-time clock, just as CLOCK0 disables it. The value of the clock can be read or assigned by using the TIME operator. ONTIME generates an interrupt when the TIME operator reaches or exceeds a specific value. ONTIME works in much the same way as the aforementioned ONEXT1 statement. Finally, the statements PH0 and PH1, when used together, ensure that outputs are given in hexadecimal notation. PH0 suppresses leading zeros, and PH1 always prints out four hexadecimal digits.



4. To function as an autopulse dialer, the Basic chip needs only a single-pole, single-throw, normally closed relay and an npn transistor to drive it.

Program 2. Placing a call with the software package

```
10   REM MCS BASIC-52 AUTO TELEPHONE DIALER
20   STRING 200,20 : REM ALLOCATE MEMORY FOR STRINGS
30   REM NAMES ARE STORED AS FOLLOWS —
40   REM    NO. OF DIGITS IN NAME — NAME —
            PHONE NUMBER
50   REM    THE "E" REPRESENTS THE END OF FILE
60   $(1) = "3JOE1234567E"
40   $(2) = "4BILL2345678E"
50   $(3) = "3ANN3456789E"
60   INPUT "WHO WOULD YOU LIKE TO DIAL —",$(4)
70   FOR I=1 TO 3 : REM LOOP TO FIND MATCH
80   REM READ THE NUMBER OF DIGITS AND STRIP OF
       ASCII BIAS
90   FOR J=1 TO (ASC($(I),1) .AND. OFH)
100  IF ASC($(4),J) = ASC($(I),J+1) THEN NEXT J ELSE NEXT I
110  CLEARS : REM CLEAR THE STACK TO ELIMINATE THE
       I INDEX
120  REM I COUNTS THE NAME, J THE PHONE NUMBER
130  CLOCK1 : REM TURN ON THE REAL TIME CLOCK
140  DO : REM NOW LOOP TO DIAL
150  C = (ASC($(I),J) .AND. OFH)
160  IF C = 0 THEN C = 10
170  FOR Y = 1 TO C : T = TIME
180  PORT1 = PORT1 .AND. OFEH : REM OPEN THE LINE
190  IF TIME <T + .04 THEN 190 : REM LOOP FOR 40 MS
200  PORT1 = PORT1 .OR. 1 : REM CLOSE THE LINE
210  IF TIME <T + .1 THEN 210 : REM LOOP FOR 60 MS
220  NEXT Y
230  J=J+1 : WHILE ASC($(I),J)<>ASC(E)
240  END
```

Building a functioning system around the Basic chip is a snap. The designer can link to virtually any standard peripheral chip simply by assigning some location to the peripheral in the 64 kbytes of external address available to the 8-bit device.

Addressing I/O

In fact, the software actually reserves the chip's upper 8 kbytes of memory (from 56 to 64 kbytes) for I/O. Accessing external peripherals is accomplished by using the XBY, or address, operator. For example:

A=XBY (0FFF0H)

reads a peripheral or external memory location that has been decoded for address $0FFF0_{16}$ and assigns its value to the variable A. Similarly:

XBY (0FFF0H) =55H

writes 55_{16} to a peripheral or memory location that has been decoded for address $0FFF0_{16}$. A system consisting of an analog-to-digital and a digital-to-analog converter plus an 8255 parallel port, all connected to the software package, is typical of process control, environmental monitoring, or energy management applications (Fig. 3).

An autopulse telephone dialer also demonstrates the versatility of the software package (Fig. 4). This application takes advantage of the logical operators as well as the real-time clock and string operations (Program 2). Such dialing could play an important role in many process control and remote environmental-monitoring systems where information must be periodically sent to a master computer over a phone link. Variations on the general software scheme could be devised to address a wide variety of automatic dialing and automatic answering settings. □

270051-8

intel

ARTICLE REPRINT

AR-409

October 1985

Increased Functions In Chip Result In Lighter, Less Costly Portable Computer

Jafar, Modares,
Application Engineer,
Intel Corporation
Chandler, Arizona

© INTEL CORPORATION, 1985

ORDER NUMBER: 270126-001

Reprinted from **Design News, 8-19-85**

270126-1

INCREASED FUNCTIONS IN CHIP RESULT IN LIGHTER, LESS COSTLY PORTABLE COMPUTER

Jafar Modares, Applications Engineer, Intel Corp., Chandler, AZ

Advances in technology have made it possible to reduce the size and increase the functionality and performance of computers and computer peripherals. With the help of microtechnology it is possible to construct a computer terminal that is smaller and lighter than a briefcase, and that can be connected to a mainframe from almost anywhere.

As more portable computers are introduced to the marketplace, the demand for lighter and even smaller systems at a lower cost are inevitable. To meet this demand changes in the architecture are necessary.

With Intel's recently introduced 80C51BH microcontroller several obstacles in the design of the portable computer have been overcome. The 80C51BH is a single chip 8-bit microcontroller that requires a single 5V power supply. It has 32 I/O lines. Its functionalities include excellent bit and byte manipulation capability at extremely high speeds as well as interfacing flexibilities through the serial and parallel channels to intelligent and unintelligent devices. It can carry its own program memory up to 4 Kbytes and has various tools and support systems.

This article discusses the implementation of a prototype portable terminal based on Intel's new 80C51BH microcontroller, and introduces the tools and techniques available to build such a computer terminal. In the application discussed, the chip monitors the keyboard, communicates with the host computer at very high BAUD rates, and displays information on the screen at slower rates for human beings. The chip also monitors the power supply for switching to the battery in case of power failure to save valuable data and computer time. The prototype is currently under futher development at Intel's Microcontroller Division.

Introducing the 80C51BH

Very new in the market, the 80C51BH is a member of Intel's MCS-51 family. The MCS-51 is a group of 8-bit microcontrollers that are extremely powerful because of their I/O structure and their bit manipulating capabilities. The 80C51BH has 4 Kbytes of on-chip program memory with the capability to address another 60 Kbytes of external program memory. In addition it has 128 bytes of on-chip RAM and can access 64 Kbytes of external data memory.

Since the 80C51BH is CMOS, it has very low power consumption (15 mA at 5V, 12 MHz). In addition, it has two power saving features not available in HMOS versions of the family. These are the Idle and Power Down modes, which are controlled by software and further reduce power comsumption. These power saving features make it ideal for battery-operated backed-up systems.

Display device

The display device of this portable terminal is a 25-line × 80-character Liquid Crystal Display (LCD). It is capable of displaying the same number of characters as a typical CRT, and it is not as bulky or heavy as the latter.

LCDs can be divided into two large categories: smart LCDs and dumb LCDs. Those displays that have the capability to receive a byte of ASCII and translate it into a displayed character are considered smart. On the other hand, dumb displays are only a matrix of dots. The former type has some kind of controller of its own plus a small memory to hold the look-up table of characters (character generator). When using a dumb display the microcontroller has to address and turn on the correct dots to make a character, this means more I/O pins will be required. However, it gives extra capabilities such as graphic displays and special or custom character generation.

Both types of displays normally accept data through an 8-bit bus. Although the LCD is relatively slow, it can still share the bus with a memory device without any degradation of the system performance.

Keyboard

Depending on their task and purpose, keyboards vary in shape, size, and the number of keypads. The keyboard for a computer terminal, as a minimum, should have all the alphanumeric keys (standard typewriter) plus a Control, an Escape, and optionally, some Function keys.

As the diagram in Figure 1 shows a

270126-2

typical keyboard consists of a matrix of eight scan lines and eight receive lines. The scan lines are connected to Port 0 of the microcontroller, and receive lines are connected to Port 1. The software writes 0s to Port 0 to hold the scan lines at a logic Low, and it writes 1s to Port 1 to hold the receive lines at a logic High. Pressing one of the keys connects a scan line to receive a line and pulls that receive line Low.

Besides being used for the scan lines, Port 0 is also the bus for the data buffer RAM and the display unit. While the controller is talking to the RAM or to the display, the bus must not be used by other devices or bus contention can occur. Since the microcontoller initiates the access to the display and to the data buffer RAM, no conflict can occur between them. However, if more than one key on the same receive line is held down simultaneously while the RAM or the display is being accessed, it is possible to foul up the information being transferred. Thus, to avoid bus contention, diodes D1 through D8 are placed on the scan lines, as shown in Figure 1.

All the receive lines are ANDed to External Interrupt 1, so that pulling any of the receive lines Low will generate an interrupt. The interrupt service routine, adapts the "two key lock-out" system and identifies the pressed key. This system allows only one key to be pressed simultaneously, all of them will be ignored.

On some keyboards, certain keys (such as Shift, Control or Escape) are not a part of the line matrix. These keys connect directly to a port pin on the microcontroller. They would not cause lock-out if pressed simultaneously with a matrix key, nor generate an interrupt if pressed singly. However, if they are part of the matrix, then the software has to recognize those keys and take proper action depending on the function of the pressed key.

Normally, when a key is pressed, the microcontroller is in the Idle mode and no other task is in progress. Pressing a key on the matrix generates an interrupt, which terminates the Idle mode. The interrupt service routine first calls a subroutine to provide a delay of approximately 25 msec (to debounce the key), and to perform the task of identifying which key is down.

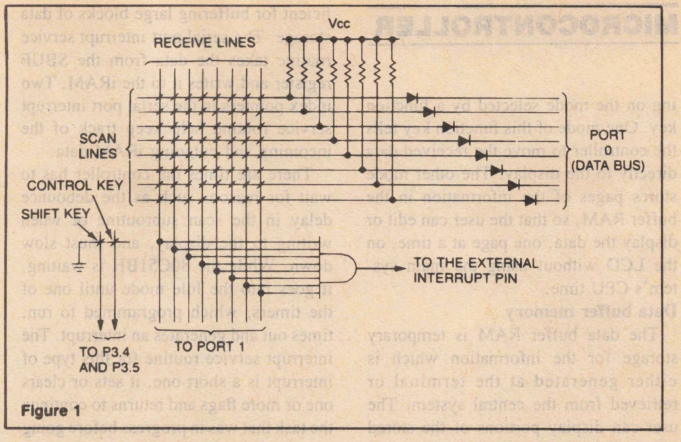


Figure 1

There are a number of ways that the interrupt service routine can identify the press key. One way is to utilize the bit-addressing capabilities of the 80C51BH to test each bit of the receive lines for a zero, which generated the interrupt, and records the bit position. Then write 0s to the receive lines, 1s to the scan lines, test the scan lines for a zero, and record its position. If the controller finds more than one zero on each port, it will discontinue any further processing and return to the main program.

Once the bit positions that contain 0 are recorded, they are used as the address for the characters in the look-up table. The subroutine finds the character that corresponds to the pressed key. The character is represented in ASCII code.

The look-up table is a list of the ASCII representation of each keypad character, and is stored in the program memory. The order in which they sit corresponds to the address generated by the scan subroutine when that character is pressed on the keyboard.

Serial communication

Once the 80C51BH has the ASCII code generated from a key closure, it can send it through its Serial Channel to the host computer. The serial port of the microcontroller can be programmed for all of the standard rates up to 375K. If the terminal is to be connected directly to the mainframe, a simple circuit translates the TTL levels to the RS232 level. The circuit also eliminates the need for the −12V supply required for RD232. The circuit diagram is shown in Figure 2.

However, the primary application of this terminal is to be the traveling person's window to the central system from any remote location. In this application a modem is needed. There are many types of modems available. Some are on PC boards for OEM use and others are ready to connect directly to the telephone by the user. They also vary in size, performance, and the way they communicate. A proper modem for the portable terminal should be small enough to carry in a briefcase and preferably be a low-power device. When an ASCII code is received by the host computer, it records that code and echoes it back to the microcontroller which displays it on the LCD.

The serial port of the 80C51BH can generate interrupts every time it receives or transmits information. The serial port interrupt must have service priority over the external interrupt generated by the keyboard. The high priority enables the serial port to receive data any time the computer addresses the terminal and transmits data.

The serial port interrupt service routine transfers the received data to the display or to a memory buffer, depend-

270126-3

MICROCONTROLLER

ing on the mode selected by a function key. One mode of this function key tells the controller to move the received data directly to the display. The other mode stores pages of the information in the buffer RAM, so that the user can edit or display the data, one page at a time, on the LCD without using the main system's CPU time.

Data buffer memory

The data buffer RAM is temporary storage for the information which is either generated at the terminal or retrieved from the central system. The user can display portions of the stored information or scroll through it. The user can also alter the data on the terminal and transmit it back to the computer in a block form.

A suitable device for this purpose is the 51C86 iRAM. This device is a pseudo-static dynamic RAM that has a built-in address latch. An internal high-speed arbitration circuit resolves any potential conflict arising between read/write and internal refresh cycles.

This iRAM is 8K × 8-bit and is sufficient for buffering large blocks of data storage. The serial port interrupt service routine takes the data from the SBUF register and writes it to the iRAM. Two index pointers in the serial port interrupt service routine help keep track of the incoming and outgoing iRAM data.

There are times the controller has to wait for reasons such as the debounce delay in the scan subroutine or when writing to the display, and must slow down. While the 80C51BH is waiting, it goes into the Idle mode until one of the timers, which programmed to run, times out and generates an interrupt. The interrupt service routine for this type of interrupt is a short one, it sets or clears one or more flags and returns to continue the task that was in progress before going into the Idle mode.

The controller is also in the Idle mode when there is no activity in the terminal, i.e., no data is being received or transmitted, and the keyboard is not being used. Therefore, is is appropriate to say that when power is applied to the terminal, the controller spends more than 90% of its time in the Idle mode.

What is the Idle mode?

When Bit 0 in the Special Function Register PCON is set, the CPU gates off its own internal clock and goes to sleep. Since the CPU consumes about 90% of the chip's power, this process saves a significant amount of power. More importantly the on chip peripherals and the RAM continue their normal functions independently of the CPU. Since the oscillator is still running, any enabled interrupt (internal or external) will wake the CPU up from its sleep, and it will start executing instructions from the interrupt service routine.

A true portable terminal should be capable of operating from a battery source. There are occasions when one would like to use the terminal at a location where an electric outlet is not readily available. But the main purpose of the battery supply is to save the data, which has been entered and stored in the buffer RAM, in the case of an unexpected power failure.

While performing all of the previously mentioned tasks, the 80C51BH can monitor its power supply, detect a power loss in its earliest stages, and initiate a power Down to save the data of the internal and external RAM.

One method for the 80C51BH to monitor its power supply is to have the positive half-cycles of the power supply transformer fed to the External Interrupt 0 pin (Figure 3). In the level activated mode, this pin generates an interrupt every time there is a high to low transition. The interrupt service routine reloads Timer 0 to a value that will make it overflow sometime between one and two periods of the line frequency. As long as the half cycles keep coming in, the timer never overflows, because it is reloaded every half a cycle. If there is a single half a cycle in which the line voltage does not reach a high enough level to generate the interrupt, the timer rolls over and generates a timer interrupt.

The interrupt service routine for Timer 0 saves the critical data of some of the internal registers and puts the controller into a Power down mode. A reset button restarts the microcontroller to resume operation when power is restored.

In this mode the CPU and all of the on-chip peripherals go to sleep. The device stops its oscillator, freezes all the

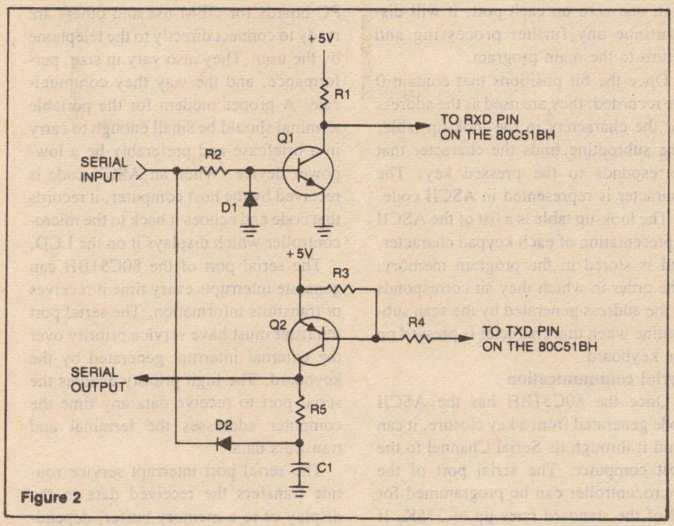


Figure 2

270126-4

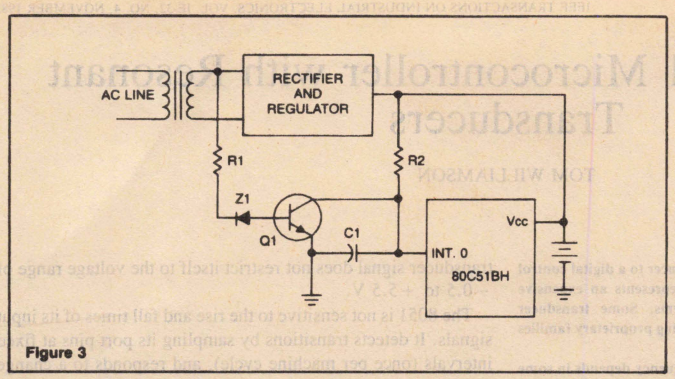


Figure 3

activities and saves the information in its internal RAM for as long as the supply voltage can be reduced to as low as 2 volts without running any risks of losing the internal RAM information. Supply current in this mode is normally 10 to 50 μA.

Bit 1 of the Special Function Register PCON controls this mode. The instruction that puts the device in the Power Down mode is the last one executed. The only way for the part to exit this mode is with a hardware reset.

A prototype of this terminal was built and connected to a MDS 800 developement system. The terminal communicated with the host computer through the serial channel at the rate of 2400 baud. The 80C51BH was emulated using Intel's ICE-51 in circuit emulator. The block diagram of the terminal is shown in Figure 4.

The CHMOS controller and LCD combination provides a system that requires only a single voltage power supply, and consumes less than 200 mW. The system's low power consumption eliminates the need for complex, voltage-regulating hardware, and cooling fans. The result is a lighter and smaller computer terminal at a very low cost. □

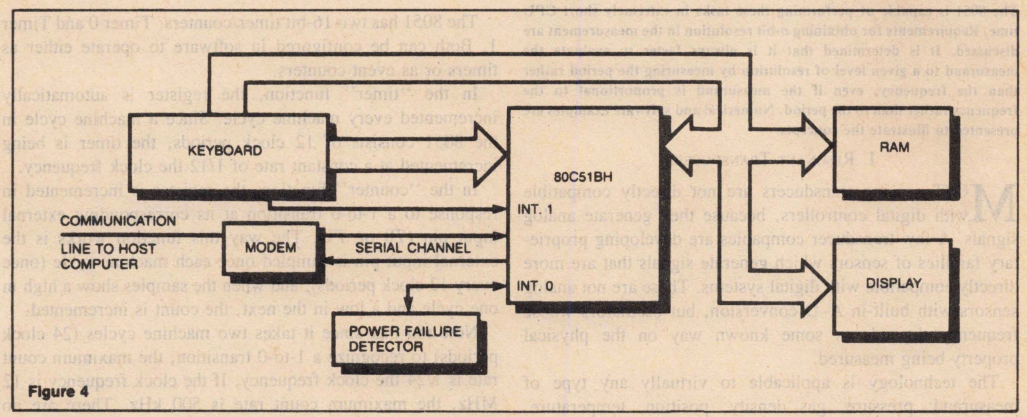


Figure 4

270126-5

IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS, VOL. IE-32, NO. 4, NOVEMBER 1985

Using the 8051 Microcontroller with Resonant Transducers

TOM WILLIAMSON

Abstract—**Having to interface an analog transducer to a digital control system through an analog-to-digital converter represents an expensive bottleneck in the development of many systems. Some transducer companies are addressing this problem by developing proprietary families of resonant transducers.**

Resonant transducers are oscillators whose frequency depends in some known way on the physical property being measured. The electrical output from these devices is a train of rectangular pulses whose repetition rate encodes the value of the measurand. Changes in the measurand cause the frequency to shift. The microcontroller detects the frequency shift, runs a validity check on it, and converts it in software to the measurand value.

This paper discusses software interfacing techniques between resonant transducers and the 8051. Techniques for measuring frequency and period are discussed and compared for resolution and interrogation time. The 8051 is capable of performing these tasks in extremely short CPU time. Requirements for obtaining *n*-bit resolution in the measurement are discussed. It is determined that it is always faster to evaluate the measurand to a given level of resolution by measuring the period rather than the frequency, even if the measurand is proportional to the frequency rather than to the period. Numerical and software examples are presented to illustrate the concepts.

I. Resonant Transducers

MOST sensing transducers are not directly compatible with digital controllers, because they generate analog signals. A few transducer companies are developing proprietary families of sensors which generate signals that are more directly compatible with digital systems. These are not analog sensors with built-in A–D conversion, but oscillators whose frequency depends in some known way on the physical property being measured.

The technology is applicable to virtually any type of measurand: pressure, gas density, position, temperature, force, etc. The sensor and microcontroller can operate from the same supply voltage, so the sensor can in most cases connect directly to a port pin on the microcontroller.

The nominal reference frequency of the output signal from these devices is in the range of 20 Hz–500 kHz, depending on the design. A change in the measurand away from the reference condition causes the frequency to shift by an amount that is related to the change in the measurand value. Transducers are available that have a full-scale frequency shift of 2–1. The microcontroller detects the change in frequency or period and converts it in software to the measurand value.

II. Connecting the Digital Transducer to the 8051

Normally the transducer output can be connected directly to one of the 8051 port pins. An exception would occur when the transducer signal does not restrict itself to the voltage range of −0.5 to +5.5 V.

The 8051 is not sensitive to the rise and fall times of its input signals. It detects transitions by sampling its port pins at fixed intervals (once per machine cycle), and responds to a change in the sequence of samples. If the slew rate of the transducer signal is extremely slow, noise superimposed on the signal could cause the sequence of samples to show false transitions. There could on that account be situations in which the transducer signal should be buffered through a Schmitt Trigger to square it up.

III. Timer/Counter Structure in the 8051

The 8051 has two 16-bit timer/counters: Timer 0 and Timer 1. Both can be configured in software to operate either as timers or as event counters.

In the "timer" function, the register is automatically incremented every machine cycle. Since a machine cycle in the 8051 consists of 12 clock periods, the timer is being incremented at a constant rate of 1/12 the clock frequency.

In the "counter" function, the register is incremented in response to a 1-to-0 transition at its corresponding external input pin (*T*0 or *T*1). The way this function works is the external input pin is sampled once each machine cycle (once every 12 clock periods), and when the samples show a high in one cycle and a low in the next, the count is incremented.

Note too that since it takes two machine cycles (24 clock periods) to recognize a 1-to-0 transition, the maximum count rate is 1/24 the clock frequency. If the clock frequency is 12 MHz, the maximum count rate is 500 kHz. There are no requirements on the duty cycle of the signal being counted.

The 8052, an enhanced version of the 8051, has three 16-bit timer/counters, two of which are identical to those in the 8051. The third timer/counter can operate either as a 16-bit timer/counter with automatic reload to a preset 16-bit value on rollover, or as a 16-bit timer/counter with a "capture" mode. In the capture mode a 1-to-0 transition at the T2EX pin causes the current value in the counting register to the "captured" into RAM. The third timer makes the 8052 particularly easy to interface with resonant transducers.

IV. Whether to Measure Frequency or Period

Measuring the frequency requires counting transducer pulses for a fixed sample time. Measuring the period requires measuring elapsed time for a fixed number of transducer pulses. For a given level of accuracy in the determination of the value of the measurand, it is usually faster to measure the period, rather than the frequency, even if the measurand is

Manuscript received October 25, 1984.
The author is with Intel Corporation, Chandler, AZ 85224.

© 1985 IEEE

proportional to frequency rather than period. However, both types of measurements will be discussed here.

Two timer/counters can be used, one to mark time and the other to count transducer pulses. If the frequency being counted does not exceed 50 kHz or so, one may equally well connect the transducer signal to an external interrupt pin, and count transducer pulses in software. That frees one timer, with very little cost in CPU time.

V. How to Measure Transducer Frequency

Measuring the frequency means counting transducer pulses for some desired sample time. The count that is directly obtained is $T \times F$, where T is the sample time and F is the frequency. The full scale range is $T \times (F\text{max} - F\text{min})$. For n-bit resolution

$$1\ \text{LSB} = \frac{T \times (F\text{max} - F\text{min})}{2^n}.$$

Therefore, the sample time required for n-bit resolution is

$$T = \frac{2^n}{F\text{max} - F\text{min}}.$$

For example, 8-bit resolution in the measurement of a frequency that varies between 5 and 10 kHz would require, according to this formula, a sample time of 51.2 ms. The maximum acceptable frequency count would be 51.2 ms × 10 kHz = 512 counts. The minimum would be 256 counts. Subtracting 256 from each frequency count would allow the frequency to be reported on a scale of 0 to *FF* in hex digits.

If *F*min and *F*max are closer together it takes more time to resolve them. 8-bit resolution in the measurement of a frequency that varies between 7 and 9 kHz would require a sample time of 128 ms. The maximum and minimum acceptable counts would be 1152 and 896. Subtracting 896 from each frequency count would allow the frequency to be reported on a scale of 0 to *FF* in hex digits.

To implement the measurement, one timer is used to establish the sample time. In this function it autoincrements every machine cycle. A machine cycle consists of 12 periods of the clock oscillator. The sample time can be converted to machine cycles by multiplying it by (*F*xtal)/12, where *F*xtal is the 8051 clock frequency. The timer needs to be preset so that it rolls over at the end of each sample time. Then it generates an interrupt, and the interrupt routine reads and clears the transducer pulse counter, and then reloads the timer with the correct preset value.

The preset or reload value is the two's complement negative of the sample time in machine cycles. For example, with a 12-MHz clock frequency, the reload value required to establish a 51.2 ms sample time is

$$-\frac{(51.2\ \text{ms}) \times (12000\ \text{kHz})}{12} = -51200 = 3800\ \text{H}.$$

In many cases the required sample time exceeds the capacity of a 16-bit timer. For example, establishing a 128 ms sample time with a 12-MHz clock frequency requires a 3-byte timer with a reload of FE0C00H. The 8051 timer, being only 2-bytes wide, can be augmented in software in the timer interrupt routine to three bytes. The 8051 has a DJNZ instruction (decrement and jump if not zero) which makes it easier to code the third timer byte to count down instead of up. If the third timer byte counts down, its reload value is the two's complement of what it would be for an up-counter. For example, if the two's complement of the sample time is FE0C00H, then the reload value for the third timer byte would be 02, instead of *FE*. The timer interrupt routine might then be

```
    DJNZ    THIRD_TIMER_BYTE,OUT
    MOV     TL0,#0
    MOV     TH0,#0CH
    MOV     THIRD_TIMER_BYTE,#02
  (Now read and clear the
    transducer pulse counter.)
OUT: RETI
```

Interrupt latency will have no effect on the measurement if the latency is the same for every sample time.

The trouble with measuring the frequency is it is not only slow, but a waste of the resolving power of the 8051's timers. A timer with microsecond resolution is being used to mark off 100-ms time periods. The technique is nevertheless useful if the timer is already serving other purposes (servicing a display, perhaps), so that the sample time is coming relatively free of charge. But in most cases it is faster and equally accurate to measure the frequency by deriving it from a measurement of the period.

VI. How to Measure the Period

Measuring the period of the transducer signal means measuring the total elapsed time over N-transducer pulses. The quantity that is directly measured is $N \times T$, where T is the period of the transducer signal in *machine cycles*. The relationship between T in machine cycles and the transducer frequency F in arbitrary frequency units is

$$T = \frac{F\text{xtal}}{F} \times (1/12)$$

where *F*xtal is the 8051 clock frequency, in the same unit as F.

The full scale range then is $N \times (T\text{max} - T\text{min})$. For n-bit resolution

$$1\ \text{LSB} = \frac{N \times (T\text{max} - T\text{min})}{2^n}.$$

Therefore, the number of periods over which the elapsed time should be measured is

$$N = \frac{2^n}{T\text{max} - T\text{min}}.$$

However, N must also be an integer. It is logical to evaluate the above formula (do not forget that *T*max and *T*min have to be in machine cycles) and select for N the next higher integer. This selection gives a period measurement that has somewhat more than n-bit resolution, which may or may not be acceptable, depending on the overall requirements of the

system. It can be scaled back to n-bit resolution, if necessary, by the following computation:

$$\text{reported value} = \frac{NT - NT\text{min}}{NT\text{max} - NT\text{min}}$$

where NT is the elapsed time measured over N periods.

The computation can be done in math if a suitable divide routine is available in the software. For 8-bit resolution it is entirely reasonable to find the reported value in a look-up table, which would take up somewhat more than one page in ROM. In fact, the look-up table would contain $NT\text{max} - NT\text{min}$ entries.

For example, suppose we want 8-bit resolution in the measurement of the period of a signal whose frequency varies from 5 to 10 kHz. If the clock frequency is 12 MHz, then $T\text{max}$ is (12 000 kHz)/(12 × 5 kHz) = 200 machine cycles, and $T\text{min}$ is 100 machine cycles. The timer needs to be on then for $N = 2.56$ periods, according to the formula. Using $N = 3$ periods will give maximum and minimum NT values of 600 and 300 machine cycles. This is somewhat more than 8-bit resolution. It can be scaled to 8 bits with a 300-byte look-up table, if desired.

To implement the measurement, one timer is used to measure the elapsed time NT. Enabling its interrupt is optional. The timer interrupt could be used to indicate a short or open in the transducer circuit.

Then the transducer is connected to one of the external interrupt pins (INT0 or INT1), and this interrupt is configured to the transition-activated mode. In the transition-activated mode every 1-to-0 transition in the transducer output will generate an interrupt. The interrupt routine counts transducer pulses, and when it gets to the predetermined N, it reads and clears the timer. For example

```
    DJNZ    PULSES,OUT
    MOV     PULSES,#PERIODS
    (Read and clear timer.)
OUT: RETI
```

If other interrupts are also to be enabled, the one connected to the transducer should be set to Priority 1, and the others to Priority 0. This is to control the interrupt response time. The response time will not affect the measurement if it is the same for every measurement. Variations in the response time will, however, affect the measurement. Setting the pulse-counter interrupt to Priority 1 and all others to Priority 0 will minimize variations in the response time. The response time will then be limited to range from 3 to 8 machine cycles.

VII. Pulsewidth Measurements

The 8051 timers have an operating mode which is particularly suited to pulsewidth measurements, and may be useful here if the transducer has a fixed duty cycle, or if the transducer output is pulsewidth modulated instead of frequency modulated by the measurand.

In this mode the timer is turned on by the on-chip circuitry in response to an input high at the external interrupt pin, and off by an input low. The external interrupt itself is enabled, so the same 1-to-0 transition from the transducer that turns off the timer also generates an interrupt. The interrupt routine would then read and reset the timer.

The advantage of this method is that the transducer signal has direct access to the timer gate, with the result that variations in the interrupt response time cease to be a factor. The timer can be read and cleared any time before the next high in the transducer output.

VIII. Deriving Frequency From a Period Measurement

We now consider the problem of measuring the transducer frequency to n-bit resolution by deriving it from a direct measurement of the period. The advantage of this technique is speed. It is always faster to measure period than frequency. But it is important to end up with a frequency value that has the same resolution and accuracy as a directly measured frequency. Two questions need to be addressed.

1) To achieve n-bit resolution in the calculated frequency, how much resolution is required in the period?
2) Having measured the period to the required resolution, what is the most efficient way to calculate the frequency?

These questions will be addressed one at a time.

IX. Resolution Requirements

In general, n-bit resolution in the frequency derivation requires somewhat more than n-bit resolution in the period measurement. How much more? It will be demonstrated presently that if the transducer frequency varies over a 2-to-1 range, the frequency can be calculated with n-bit resolution from period measurements that have $(n + 1)$-bit resolution.

The more practical form of the question is over how many periods (N) must the transducer signal be sampled to obtain the required resolution in F? And so, we commence a calculation of N.

The basic calculation of frequency from $N \times T$ (which we shall call NT) is straightforward:

$$F = N/(NT).$$

The relationship between an increment dF in the calculated frequency due to an increment $d(NT)$ in the measured period is, therefore,

$$dF = -\frac{N}{(NT)^2}\, d(NT)$$

$$= -\frac{F^2}{N}\, d(NT).$$

This equation says the value of an LSB in the calculated frequency is $(F^2)/N \times$ the value of an LSB in NT. Then the maximum value that an LSB in the calculated frequency can have is $(F\text{max})^2/N \times$ the value of an LSB in NT. For the calculated frequency to have n-bit resolution over the entire range of frequencies, the value of its LSB must never exceed $(F\text{max} - F\text{min})/2^n$. Therefore, the measurement requires

$$\frac{(F\text{max})^2}{N} \times (1 \text{ LSB in } NT) \leq \frac{F\text{max} - F\text{min}}{2^n}.$$

The required resolution in NT is, therefore,

$$1 \text{ LSB in } NT \leq \frac{N \times (F\max - F\min)}{2^n \times (F\max)^2}.$$

Now, to say that NT is measured with m-bit resolution means

$$1 \text{ LSB in } NT = \frac{N \times (1/F\min - 1/F\max)}{2^m}.$$

Substituting this value for 1 LSB into the required resolution and solving for 2^m yields

$$2^m \geq \frac{F\max}{F\min} \times 2^n.$$

Then the requirement on m is

$$m \geq n + \frac{\ln (F\max/F\min)}{\ln (2)}.$$

It can be stated with some certainty, then, that if the transducer frequency varies over a range of 2-to-1, the frequency can be calculated with 8-bit resolution from a period measurement that has 9-bit resolution. If the frequency variation is less than 2-to-1, another full bit of resolution in the period measurement is not needed.

To obtain m-bit resolution in NT, N must satisfy

$$N \geq \frac{2^m}{T\max - T\min}.$$

Substituting for 2^m, and using $T\max = 1/F\min$ and $T\min = 1/F\max$, gives the result that

$$N \geq \frac{(F\max)^2}{F\max - F\min} \times 2^n.$$

It should be noted that the units of frequency here are periods/machine cycle, since the 8051 measures time by counting machine cycles. The conversion factor between Hz and periods/machine cycle is 12/(clock frequency). So the requirement on N can also be written

$$N \geq \frac{F\max}{F\max - F\min} \times \frac{F\max}{F\text{xtal}} \times 12 \times 2^n$$

where Fxtal is the 8051 clock frequency in the same units as Fmax and Fmin. This is the number of transducer pulses over which the transducer signal must be sampled to achieve the required solution in F.

For example, suppose that 8-bit resolution is required in F, where Fmax = 10 kHz and Fmin = 5 kHz, and that Fxtal = 12 MHz. Then the above calculation shows that $N = 6$ periods gives sufficient resolution in the period measurement to satisfy the resolution requirement in F. Six periods will take 0.6–1.2 ms of sampling time, on that frequency range. Recall that the sample time for a direct frequency measurement of the same signal and to the same resolution was earlier calculated to be 51.2 ms.

X. Computing the Frequency from the Period

The period measurement leaves one with a 16-bit integer, which is $N \times T$ (or NT) in machine cycles. The conversion to frequency is straightforward:

$$F = N/(NT) \text{ periods/machine cycle.}$$

The quantity of interest is probably not F, but a normalized measure of the amount by which F exceeds its minimum acceptable value. This quantity represents, through the transducer's transfer function, the "reported value" of the measurand, and this quantity is an n-bit integer whose value ranges from 0 (all bits = 0) to full scale (all bits = 1). This normalized frequency is

$$f = \frac{F - F\min}{F\max - F\min}$$

$$= \frac{F\min}{F\max - F\min} \times (F/F\min - 1)$$

Using $F = N/(NT)$ and F min $= N/(NT$ max) allows the normalized frequency to be written

$$f = \frac{F\min}{F\max - F\min} \times \frac{NT\max - NT}{NT}.$$

To get a handle on what kinds of numbers are involved here, consider the situation where 8-bit resolution is required in f, and in which Fxtal = 12 MHz, Fmax = 10 kHz, and Fmin = 5 kHz. We have previously determined that for this set of conditions, $N = 6$ periods gives sufficient resolution in the period measurement to satisfy the resolution requirement in F (and in f). With a 12 MHz clock frequency, Tmax in machine cycles is (12 000 kHz)/(12 × 5 kHz) = 200 machine cycles, so NTmax is 6 × 200 = 1200 machine cycles. The calculation for f then becomes

$$f = \frac{1200 - NT}{NT}.$$

The minimum acceptable value that NT can have is ($N \times T$min + 1), where Tmin = (12 000 kHz)/(12 × 10 kHz) = 100 machine cycles. Then $N \times T$min = 6 × 100 = 600 machine cycles. The allowable values for NT are then 601–1200 machine cycles, a total of 599 different values.

The fastest way to "calculate" f would be with a 599-byte look-up table. This method has the added advantage that nonlinearities in the transfer function between frequency and measurand can be built into the table. Look-up tables are facilitated in the 8051 by the MOVC A,@A + PC, and MOVC A,@A + DPTR instructions. DPTR is a 16-bit "data pointer" register in the 8051. Its two bytes can be individually addressed as DPL (low byte) and DPH (high byte).

In the example under discussion, it will be necessary to load DPTR with the address of the first byte of the look-up table, less 601, plus the 2-byte value of NT. The software that accesses the table might then take the following form:

```
TABLE EQU (address of first table entry)
```

```
;                          DIVIDE ROUTINE
;
; This divide routine calculates
;
;                            numerator
;              quotient = ------------
;                            denominator
;
; in which numerator and denominator are integers and
; numerator < denominator.  Quotient is of the form
;
;              quotient = 0.  q1 q2 q3 . . . qN
;
; where qn is the coefficient of 2**(-n).  The procedure is
;
;     numerator: = 2 x numerator
;     n: = 1
;     while n <= N+1 do
;          if numerator >= denominator then qn: = 0 else qn: = 1
;          if qn = 0 then numerator: = 2 x numerator
;                    else numerator: = 2 x numerator - denominator
;          increment n
;     end_while
;
```

Fig. 1. A divide algorithm.

```
MOV     A,#LOW(TABLE-601)
ADD     A,NT_LO
MOV     DPL,A
MOV     A,#HIGH(TABLE-601)
ADDC    A,NT_HI
MOV     DPH,A
CLR     A
MOVC    A,@A+DPTR.
```

At this point the accumulator contains the 8-bit value of f.

It is perfectly reasonable to decide that a 599-byte look-up table is unwieldy. Its advantages are speed and built-in error correction. But a reasonably fast divide algorithm can be written to this specific purpose, making use of *a priori* knowledge about the sizes of the numbers that are involved in the computation. It helps to know that in this example the numerator is never going to be larger than 599 and the denominator is always greater than the numerator.

A complete discussion of divide routines is beyond the scope of this paper, but a suitable divide algorithm for this specific application is shown in Fig. 1. Reference [1] calls this the Restoring division algorithm. It is particularly well suited to the 8051, because "<" comparisons are greatly facilitated by the 8051's CJNE (compare and jump if not equal) instruction. CJNE A,B,rel, executes a relative jump if A does not equal B. More importantly to this application, the instruction sets the carry flag if $A < B$.

XI. Accuracy and Resolution

The accuracy with which the 8051 will measure the frequency or period of the transducer signal depends on two things: the accuracy of the clock oscillator and variations in the interrupt response time.

Since the clock signal is normally generated by a crystal oscillator, the oscillator accuracy normally far exceeds the quantizing error inherent in the finite (n-bit) resolution.

As was previously mentioned, interrupt response time does not introduce an error into the measurement itself, but variations in the interrupt response time can. Interrupt response time in the 8051 can vary from 3 to 8 machine cycles, depending on what instruction is in progress at the time the interrupt is generated. This would represent an error of ± 5 counts in the measured value of NT during a period measurement. An error of ± 5 counts in NT does not necessarily translate to ± 5 LSB's in the final result, but it might still represent an error that exceeds the resolution.

In a direct frequency measurement variations in the interrupt response time would represent an error of ± 5 μs in the sample time.

If these kinds of errors are unacceptable there are ways to deal with them. In period measurements, if the duty cycle of the transducer is constant, the pulsewidth measurement technique, previously described, can be used. Its advantage is that it gates the timer off when the interrupt is generated, rather than when the interrupt is responded to.

In other cases one can simply increase the sample time above the minimum required to obtain the desired resolution. For example, if the measurement requires 8-bit resolution, one can design the software for an 11-bit resolution and truncate the result to 8 bits.

References

[1] Davio *et al., Digital Systems with Algorithm Implementation.* New York: Wiley, 1983.

MCS®-80/85 Data Sheets

11

8080A/8080A-1/8080A-2
8-BIT N-CHANNEL MICROPROCESSOR

- **TTL Drive Capability**
- **2 μs (−1:1.3 μs, −2:1.5 μs) Instruction Cycle**
- **Powerful Problem Solving Instruction Set**
- **6 General Purpose Registers and an Accumulator**
- **16-Bit Program Counter for Directly Addressing up to 64K Bytes of Memory**
- **16-Bit Stack Pointer and Stack Manipulation Instructions for Rapid Switching of the Program Environment**
- **Decimal, Binary, and Double Precision Arithmetic**
- **Ability to Provide Priority Vectored Interrupts**
- **512 Directly Addressed I/O Ports**
- **Available in EXPRESS — Standard Temperature Range**
- **Available in 40-Lead Cerdip and Plastic Packages**
  (See Packaging Spec. Order #231369)

The Intel® 8080A is a complete 8-bit parallel central processing unit (CPU). It is fabricated on a single LSI chip using Intel's n-channel silicon gate MOS process. This offers the user a high performance solution to control and processing applications.

The 8080A contains 6 8-bit general purpose working registers and an accumulator. The 6 general purpose registers may be addressed individually or in pairs providing both single and double precision operators. Arithmetic and logical instructions set or reset 4 testable flags. A fifth flag provides decimal arithmetic operation.

The 8080A has an external stack feature wherein any portion of memory may be used as a last in/first out stack to store/retrieve the contents of the accumulator, flags, program counter, and all of the 6 general purpose registers. The 16-bit stack pointer controls the addressing of this external stack. This stack gives the 8080A the ability to easily handle multiple level priority interrupts by rapidly storing and restoring processor status. It also provides almost unlimited subroutine nesting.

This microprocessor has been designed to simplify systems design. Separate 16-line address and 8-line bidirectional data busses are used to facilitate easy interface to memory and I/O. Signals to control the interface to memory and I/O are provided directly by the 8080A. Ultimate control of the address and data busses resides with the HOLD signal. It provides the ability to suspend processor operation and force the address and data busses into a high impedance state. This permits OR-tying these busses with other controlling devices for (DMA) direct memory access or multi-processor operation.

NOTE:
The 8080A is functionally and electrically compatible with the Intel 8080.

November 1986
Order Number: 231453-001

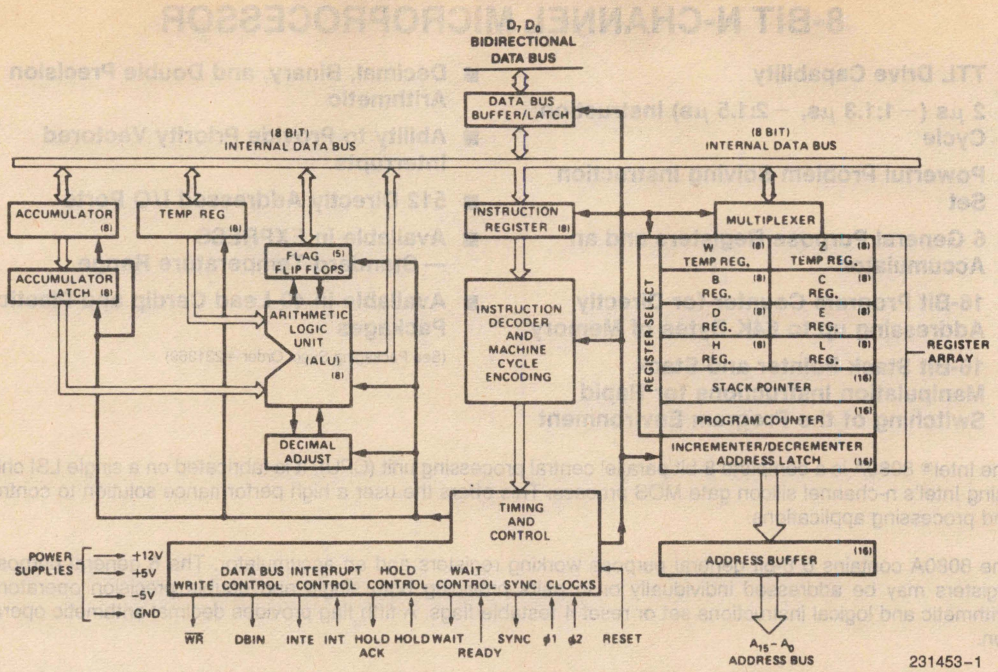
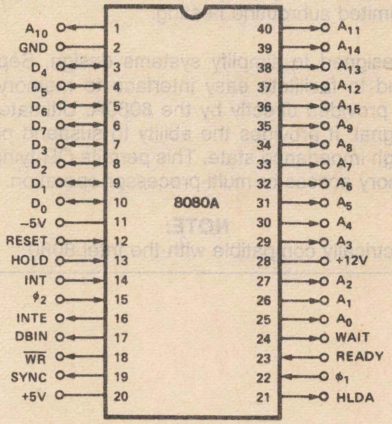


Figure 1. Block Diagram



231453-2

Figure 2. Pin Configuration

Table 1. Pin Description

Symbol	Type	Name and Function
A_{15}–A_0	O	**ADDRESS BUS:** The address bus provides the address to memory (up to 64K 8-bit words) or denotes the I/O device number for up to 256 input and 256 output devices. A_0 is the least significant address bit.
D_7–D_0	I/O	**DATA BUS:** The data bus provides bi-directional communication between the CPU, memory, and I/O devices for instructions and data transfers. Also, during the first clock cycle of each machine cycle, the 8080A outputs a status word on the data bus that describes the current machine cycle. D_0 is the least significant bit.
SYNC	O	**SYNCHRONIZING SIGNAL:** The SYNC pin provides a signal to indicate the beginning of each machine cycle.
DBIN	O	**DATA BUS IN:** The DBIN signal indicates to external circuits that the data bus is in the input mode. This signal should be used to enable the gating of data onto the 8080A data bus from memory or I/O.
READY	I	**READY:** The READY signal indicates to the 8080A that valid memory or input data is available on the 8080A data bus. This signal is used to synchronize the CPU with slower memory or I/O devices. If after sending an address out the 8080A does not receive a READY input, the 8080A will enter a WAIT state for as long as the READY line is low. READY can also be used to single step the CPU.
WAIT	O	**WAIT:** The WAIT signal acknowledges that the CPU is in a WAIT state.
$\overline{WR}$	O	**WRITE:** The $\overline{WR}$ signal is used for memory WRITE or I/O output control. The data on the data bus is stable while the $\overline{WR}$ signal is active low ($\overline{WR} = 0$).
HOLD	I	**HOLD:** The HOLD signal requests the CPU to enter the HOLD state. The HOLD state allows an external device to gain control of the 8080A address and data bus as soon as the 8080A has completed its use of these busses for the current machine cycle. It is recognized under the following conditions: • the CPU is in the HALT state. • the CPU is in the T2 or TW state and the READY signal is active. As a result of entering the HOLD state the CPU ADDRESS BUS (A_{15}–A_0) and DATA BUS (D_7–D_0) will be in their high impedance state. The CPU acknowledges its state with the HOLD ACKNOWLEDGE (HLDA) pin.
HLDA	O	**HOLD ACKNOWLEDGE:** The HLDA signal appears in response to the HOLD signal and indicates that the data and address bus will go to the high impedance state. The HLDA signal begins at: • T3 for READ memory or input. • The Clock Period following T3 for WRITE memory or OUTPUT operation. In either case, the HLDA signal appears after the rising edge of ϕ_2.
INTE	O	**INTERRUPT ENABLE:** Indicates the content of the internal interrupt enable flip/flop. This flip/flop may be set or reset by the Enable and Disable Interrupt instructions and inhibits interrupts from being accepted by the CPU when it is reset. It is automatically reset (disabling further interrupts) at time T1 of the instruction fetch cycle (M1) when an interrupt is accepted and is also reset by the RESET signal.
INT	I	**INTERRUPT REQUEST:** The CPU recognizes an interrupt request on this line at the end of the current instruction or while halted. If the CPU is in the HOLD state or if the Interrupt Enable flip/flop is reset it will not honor the request.
RESET[1]	I	**RESET:** While the RESET signal is activated, the content of the program counter is cleared. After RESET, the program will start at location 0 in memory. The INTE and HLDA flip/flops are also reset. Note that the flags, accumulator, stack pointer, and registers are not cleared.
V_{SS}		**GROUND:** Reference.
V_{DD}		**POWER:** +12 ±5% V.
V_{CC}		**POWER:** +5 ±5% V.
V_{BB}		**POWER:** −5 ±5% V.
ϕ_1, ϕ_2		**CLOCK PHASES:** 2 externally supplied clock phases. (non TTL compatible)

NOTE:
1. The RESET signal must be active for a minimum of 3 clock cycles.

ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias ..... 0°C to +70°C
Storage Temperature ..... −65°C to +150°C
All Input or Output Voltages with Respect to V_{BB} ..... −0.3V to +20V
V_{CC}, V_{DD} and V_{SS} with Respect to V_{BB} ..... −0.3V to +20V
Power Dissipation 1.5W

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS

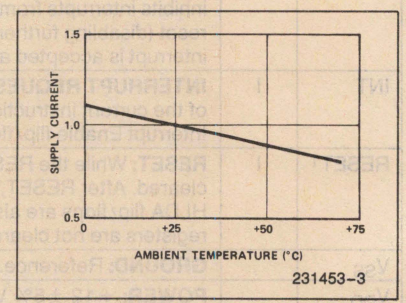
T_A = 0°C to 70°C, V_{DD} = +12V ±5%, V_{CC} = +5V ±5%, V_{BB} = −5V ±5%, V_{SS} = 0V; unless otherwise noted

Symbol	Parameter	Min	Typ	Max	Unit	Test Condition
V_{ILC}	Clock Input Low Voltage	V_{SS} − 1		V_{SS} + 0.8	V	
V_{IHC}	Clock Input High Voltage	9.0		V_{DD} + 1	V	
V_{IL}	Input Low Voltage	V_{SS} − 1		V_{SS} + 0.8	V	
V_{IH}	Input High Voltage	3.3		V_{CC} + 1	V	
V_{OL}	Output Low Voltage			0.45	V	I_{OL} = 1.9 mA on All Outputs, I_{OH} = −150 μA.
V_{OH}	Output High Voltage	3.7			V	
$I_{DD\ (AV)}$	Avg. Power Supply Current (V_{DD})		40	70	mA	Operation T_{CY} = 0.48 μs
$I_{CC\ (AV)}$	Avg. Power Supply Current (V_{CC})		60	80	mA	
$I_{BB\ (AV)}$	Avg. Power Supply Current (V_{BB})		0.01	1	mA	
I_{IL}	Input Leakage			±10	μA	$V_{SS} \le V_{IN} \le V_{CC}$
I_{CL}	Clock Leakage			±10	μA	$V_{SS} \le V_{CLOCK} \le V_{DD}$
I_{DL}	Data Bus Leakage in Input Mode			−100 −2.0	μA mA	$V_{SS} \le V_{IN} \le V_{SS}$ + 0.8V V_{SS} + 0.8V $\le V_{IN} \le V_{CC}$
I_{FL}	Address and Data Bus Leakage During HOLD			+10 −100	μA	$V_{ADDR/DATA} = V_{CC}$ $V_{ADDR/DATA} = V_{SS}$ + 0.45V

CAPACITANCE

T_A = 25°C, V_{CC} = V_{DD} = V_{SS} = 0V, V_{BB} = −5V

Symbol	Parameter	Typ	Max	Unit	Test Condition
C_ϕ	Clock Capacitance	17	25	pF	f_c = 1 MHz
C_{IN}	Input Capacitance	6	10	pF	Unmeasured Pins
C_{OUT}	Output Capacitance	10	20	pF	Returned to V_{SS}

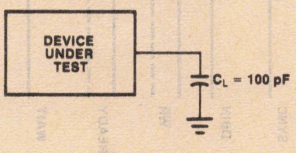


Typical Supply Current vs Temperature, Normalized
ΔI Supply/ΔT_A = −0.45%/°C

A.C. CHARACTERISTICS (8080A) T_A = 0°C to 70°C, V_{DD} = +12V ±5%, V_{CC} = +5V ±5%, V_{BB} = −5V ±5%, V_{SS} = 0V; unless otherwise noted

Symbol	Parameter	Min	Max	−1 Min	−1 Max	−2 Min	−2 Max	Unit	Test Condition
t_{CY}(3)	Clock Period	0.48	2.0	0.32	2.0	0.38	2.0	μs	
t_r, t_f	Clock Rise and Fall Time	0	50	0	25	0	50	ns	
$t_{\phi 1}$	ϕ_1 Pulse Width	60		50		60		ns	
$t_{\phi 2}$	ϕ_2 Pulse Width	220		145		175		ns	
t_{D1}	Delay ϕ_1 to ϕ_2	0		0		0		ns	
t_{D2}	Delay ϕ_1 to ϕ_2	70		60		70		ns	
t_{D3}	Delay ϕ_1 to ϕ_2 Leading Edges	80		60		70	ns		
t_{DA}	Address Output Delay From ϕ_2		200		150		175	ns	C_L = 100 pF
t_{DD}	Data Output Delay From ϕ_2		200		180		200	ns	
t_{DC}	Signal Output Delay From ϕ_1 or ϕ_2 (SYNC, WR, WAIT, HLDA)		120		110		120	ns	C_L = 50 pF
t_{DF}	DBIN Delay From ϕ_2	25	140	25	130	25	140	ns	
t_{DI}(1)	Delay for Input Bus to Enter Input Mode		t_{DF}		t_{DF}		t_{DF}	ns	
t_{DS1}	Data Setup Time During ϕ_1 and DBIN	30		10		20		ns	
t_{DS2}	Data Setup Time to ϕ_2 During DBIN	150		120		130		ns	
t_{DH}(1)	Data Hold Time From ϕ_2 and DBIN	(1)		(1)		(1)		ns	
t_{IE}	INTE Output Delay From ϕ_2		200		200		200	ns	C_L = 50 pF
t_{RS}	READY Setup Time During ϕ_2	120		90		90		ns	
t_{HS}	HOLD Setup Time During ϕ_2	140		120		120		ns	
t_{IS}	INT Setup Time During ϕ_2	120		100		100		ns	
t_H	Hold Time From ϕ_2 (READY, INT, HOLD)	0		0		0		ns	
t_{FD}	Delay to Float During Hold (Address and Data Bus)		120		120		120	ns	
t_{AW}	Address Stable Prior to WR	(5)		(5)		(5)		ns	
t_{DW}	Output Data Stable Prior to WR	(6)		(6)		(6)		ns	
t_{WD}	Output Data Stable From WR	(7)		(7)		(7)		ns	
t_{WA}	Address Stable From WR	(7)		(7)		(7)		ns	
t_{HF}	HLDA to Float Delay	(8)		(8)		(8)		ns	
t_{WF}	WR to Float Delay	(9)		(9)		(9)		ns	
t_{AH}	Address Hold Time After DBIN During HLDA	−20		−20		−20		ns	

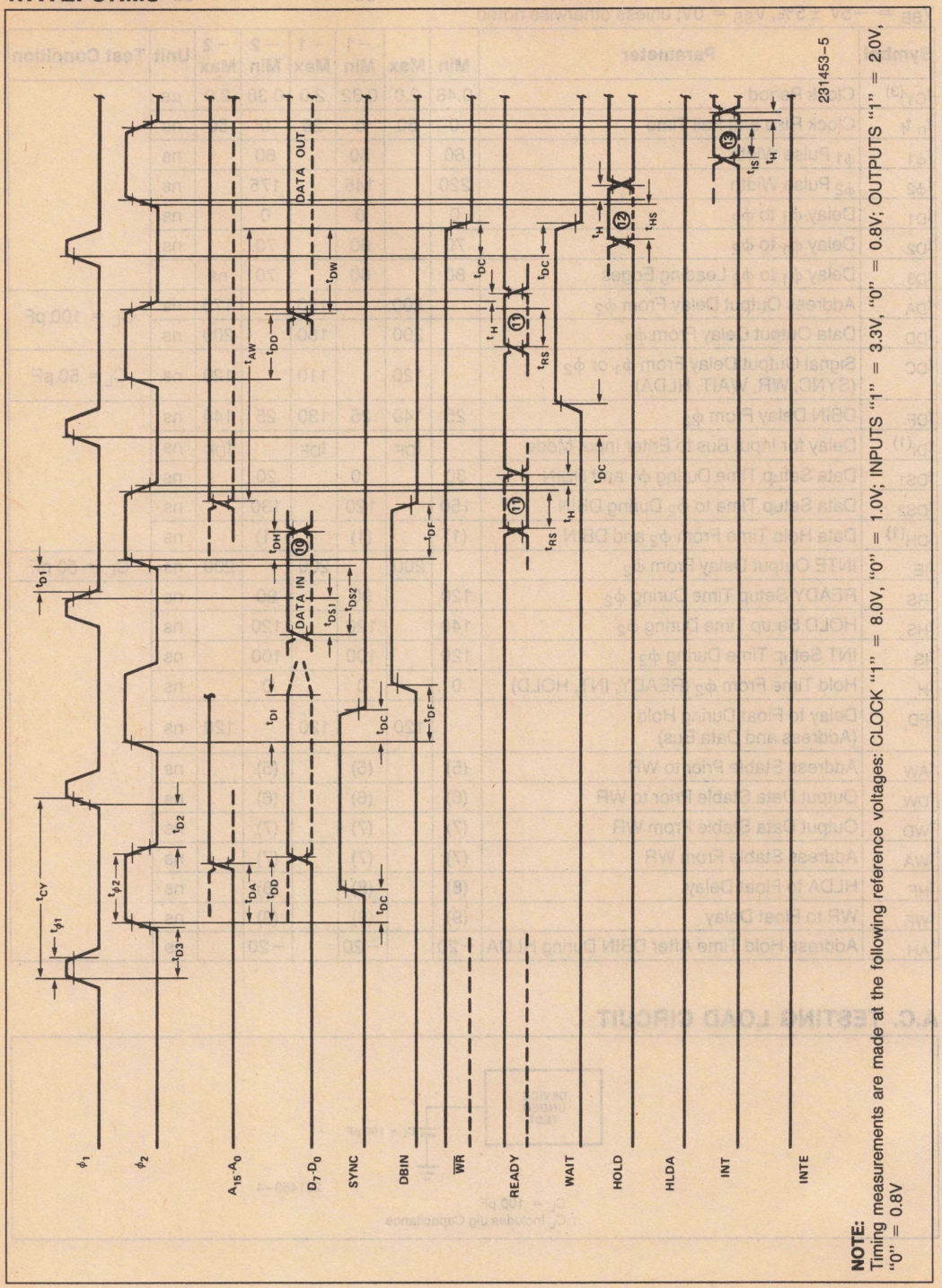
A.C. TESTING LOAD CIRCUIT



231453-4

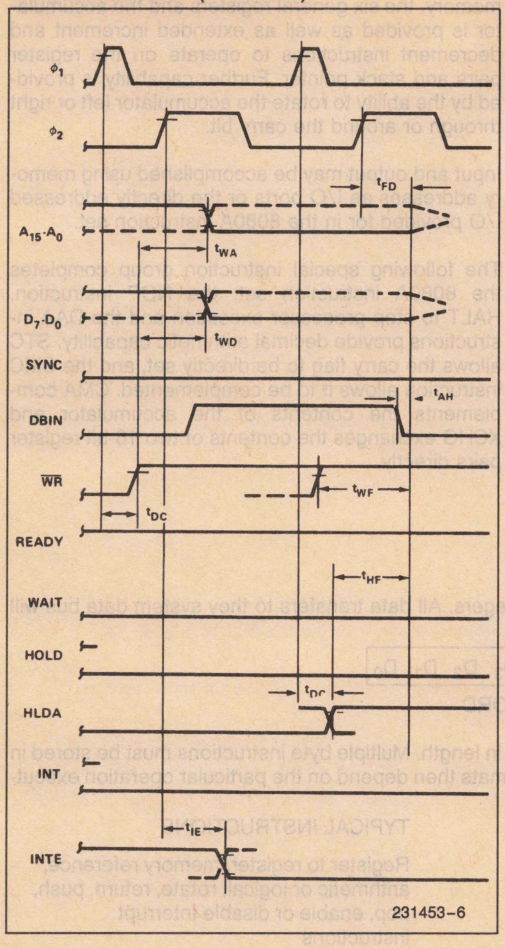
C_L = 100 pF
C_L Includes Jig Capacitance

WAVEFORMS



NOTE:
Timing measurements are made at the following reference voltages: CLOCK "1" = 8.0V, "0" = 1.0V; INPUTS "1" = 3.3V, "0" = 0.8V; OUTPUTS "1" = 2.0V, "0" = 0.8V

WAVEFORMS (Continued)

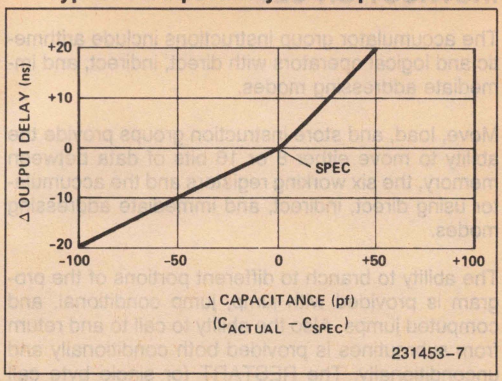


NOTES:

(Parenthesis gives −1, −2 specifications, respectively.)

1. Data input should be enabled with DBIN status. No bus conflict can then occur and data hold time is assured.
t_{DH} = 50 ns or t_{DF}, whichever is less.
2. $t_{CY} = t_{D3} + t_{r\phi2} + t_{\phi2} + t_{f\phi2} + t_{D2} + t_{r\phi1} \geq$ 480 ns (−1:320 ns, − 2:380 ns).

Typical Δ Output Delay vs Δ Capacitance



3. The following are relevant when interfacing the 8080A to devices having V_{IH} = 3.3V:
   a) Maximum output rise time from 0.8V to 3.3V = 100 ns @ C_L = SPEC.
   b) Output delay when measured to 3.0V = SPEC + 60 ns @ C_L = SPEC.
   c) If C_L = SPEC, add 0.6 ns/pF if $C_L > C_{SPEC}$, subtract 0.3 ns/pF (from modified delay) if $C_L < C_{SPEC}$.
4. $t_{AW} = 2\,t_{CY} - t_{D3} - t_{r\phi2} - 140$ ns (−1:110 ns, − 2:130 ns).
5. $t_{DW} = t_{CY} - t_{D3} - t_{r\phi2} - 170$ ns (−1:150 ns, − 2:170 ns).
6. If not HLDA, $t_{WD} = t_{WA} = t_{D3} + t_{r\phi2} + 10$ ns. If HLDA, $t_{WD} = t_{WA} = t_{WF}$.
7. $t_{HF} = t_{D3} + t_{r\phi2} - 50$ ns.
8. $t_{WF} = t_{D3} + t_{r\phi2} - 10$ ns.
9. Data in must be stable for this period during DBIN T_3. Both t_{DS1} and t_{DS2} must be satisfied.
10. Ready signal must be stable for this period during T_2 or T_W. (Must be externally synchronized.)
11. Hold signal must be stable for this period during T_2 or T_W when entering hold mode, and during T_3, T_4, T_5 and T_{WH} when in hold mode. (External synchronization is not required.)
12. Interrupt signal must be stable during this period of the last clock cycle of any instruction in order to be recognized on the following instruction. (External synchronization is not required.)
13. This timing diagram shows timing relationships only; it does not represent any specific machine cycle.

INSTRUCTION SET

The accumulator group instructions include arithmetic and logical operators with direct, indirect, and immediate addressing modes.

Move, load, and store instruction groups provide the ability to move either 8 or 16 bits of data between memory, the six working registers and the accumulator using direct, indirect, and immediate addressing modes.

The ability to branch to different portions of the program is provided with jump, jump conditional, and computed jumps. Also the ability to call to and return from subroutines is provided both conditionally and unconditionally. The RESTART (or single byte call instruction) is useful for interrupt vector operation.

Double precision operators such as stack manipulation and double add instructions extend both the arithmetic and interrupt handling capability of the 8080A. The ability to increment and decrement memory, the six general registers and the accumulator is provided as well as extended increment and decrement instructions to operate on the register pairs and stack pointer. Further capability is provided by the ability to rotate the accumulator left or right through or around the carry bit.

Input and output may be accomplished using memory addresses as I/O ports or the directly addressed I/O provided for in the 8080A instruction set.

The following special instruction group completes the 8080A instruction set: the NOP instruction, HALT to stop processor execution and the DAA instructions provide decimal arithmetic capability. STC allows the carry flag to be directly set, and the CMC instruction allows it to be complemented. CMA complements the contents of the accumulator and XCHG exchanges the contents of two 16-bit register pairs directly.

Data and Instruction Formats

Data in the 8080A is stored in the form of 8-bit binary integers. All data transfers to they system data bus will be in the same format.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0

DATA WORD

The program instructions may be one, two, or three bytes in length. Multiple byte instructions must be stored in successive words in program memory. The instruction formats then depend on the particular operation executed.

		TYPICAL INSTRUCTIONS
One Byte Instructions		
D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0	OP CODE	Register to register, memory reference, arithmetic or logical, rotate, return, push, pop, enable or disable Interrupt instructions
Two Byte Instructions		
D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0	OP CODE	Immediate mode or I/O instructions
D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0	OPERAND	
Three Byte Instructions		
D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0	OP CODE	Jump, call or direct load and store instructions
D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0	LOW ADDRESS OR OPERAND 1	
D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0	HIGH ADDRESS OR OPERAND 2	

For the 8080A a logic "1" is defined as a high level and a logic "0" is defined as a low level.

Table 2. Instruction Set Summary

Mnemonic*	Instruction Code (1) D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0	Operations Description	Clock Cycles (2)
MOVE, LOAD, AND STORE			
MOVr1,r2	0 1 D D D S S S	Move register to register	5
MOV M,r	0 1 1 1 0 S S S	Move register to memory	7
MOV r,M	0 1 D D D 1 1 0	Move memory to register	7
MVI r	0 0 D D D 1 1 0	Move immediate register	7
MVI M	0 0 1 1 0 1 1 0	Move immediate memory	10
LXI B	0 0 0 0 0 0 0 1	Load immediate register Pair B & C	10
LXI D	0 0 0 1 0 0 0 1	Load immediate register Pair D & E	10
LXI H	0 0 1 0 0 0 0 1	Load immediate register Pair H & L	10
STAX B	0 0 0 0 0 0 1 0	Store A indirect	7
STAX D	0 0 0 1 0 0 1 0	Store A indirect	7
LDAX B	0 0 0 0 1 0 1 0	Load A indirect	7
LDAX D	0 0 0 1 1 0 1 0	Load A indirect	7
STA	0 0 1 1 0 0 1 0	Store A direct	13
LDA	0 0 1 1 1 0 1 0	Load A direct	13
SHLD	0 0 1 0 0 0 1 0	Store H & L direct	16
LHLD	0 0 1 0 1 0 1 0	Load H & L direct	16
XCHG	1 1 1 0 1 0 1 1	Exchange D & E, H & L Registers	4
STACK OPS			
PUSH B	1 1 0 0 0 1 0 1	Push register Pair B & C on stack	11
PUSH D	1 1 0 1 0 1 0 1	Push register Pair D & E on stack	11
PUSH H	1 1 1 0 0 1 0 1	Push register Pair H & L on stack	11
PUSH PSW	1 1 1 1 0 1 0 1	Push A and Flags on stack	11
POP B	1 1 0 0 0 0 0 1	Pop register Pair B & C off stack	10
POP D	1 1 0 1 0 0 0 1	Pop register Pair D & E off stack	10
POP H	1 1 1 0 0 0 0 1	Pop register Pair H & L off stack	10
POP PSW	1 1 1 1 0 0 0 1	Pop A and Flags off stack	10
XTHL	1 1 1 0 0 0 1 1	Exchange top of stack, H & L	18
SPHL	1 1 1 1 1 0 0 1	H & L to stack pointer	5
LXI SP	0 0 1 1 0 0 0 1	Load immediate stack pointer	10
INX SP	0 0 1 1 0 0 1 1	Increment stack pointer	5
DCX SP	0 0 1 1 1 0 1 1	Decrement stack pointer	5
JUMP			
JMP	1 1 0 0 0 0 1 1	Jump unconditional	10
JC	1 1 0 1 1 0 1 0	Jump on carry	10
JNC	1 1 0 1 0 0 1 0	Jump on no carry	10
JZ	1 1 0 0 1 0 1 0	Jump on zero	10
JNZ	1 1 0 0 0 0 1 0	Jump on no zero	10
JP	1 1 1 1 0 0 1 0	Jump on positive	10
JM	1 1 1 1 1 0 1 0	Jump on minus	10
JPE	1 1 1 0 1 0 1 0	Jump on parity even	10
JPO	1 1 1 0 0 0 1 0	Jump on parity odd	10
PCHL	1 1 1 0 1 0 0 1	H & L to program counter	5
CALL			
CALL	1 1 0 0 1 1 0 1	Call unconditional	17
CC	1 1 0 1 1 1 0 0	Call on carry	11/17
CNC	1 1 0 1 0 1 0 0	Call on no carry	11/17
CZ	1 1 0 0 1 1 0 0	Call on zero	11/17
CNZ	1 1 0 0 0 1 0 0	Call on no zero	11/17
CP	1 1 1 1 0 1 0 0	Call on positive	11/17
CM	1 1 1 1 1 1 0 0	Call on minus	11/17
CPE	1 1 1 0 1 1 0 0	Call on parity even	11/17
CPO	1 1 1 0 0 1 0 0	Call on parity odd	11/17
RETURN			
RET	1 1 0 0 1 0 0 1	Return	10
RC	1 1 0 1 1 0 0 0	Return on carry	5/11
RNC	1 1 0 1 0 0 0 0	Return on no carry	5/11
RZ	1 1 0 0 1 0 0 0	Return on zero	5/11
RNZ	1 1 0 0 0 0 0 0	Return on no zero	5/11
RP	1 1 1 1 0 0 0 0	Return on positive	5/11
RM	1 1 1 1 1 0 0 0	Return on minus	5/11
RPE	1 1 1 0 0 0 0 0	Return on parity even	5/11
RPO	1 1 1 0 0 0 0 0	Return on parity odd	5/11
RESTART			
RST	1 1 A A A 1 1 1	Restart	11
INCREMENT AND DECREMENT			
INR r	0 0 D D D 1 0 0	Increment register	5
DCR r	0 0 D D D 1 0 1	Decrement register	5
INR M	0 0 1 1 0 1 0 0	Increment memory	10
DCR M	0 0 1 1 0 1 0 1	Decrement memory	10
INX B	0 0 0 0 0 0 1 1	Increment B & C registers	5
INX D	0 0 0 1 0 0 1 1	Increment D & E registers	5
INX H	0 0 1 0 0 0 1 1	Increment H & L registers	5
DCX B	0 0 0 0 1 0 1 1	Decrement B & C	5
DCX D	0 0 0 1 1 0 1 1	Decrement D & E	5
DCX H	0 0 1 0 1 0 1 1	Decrement H & L	5
ADD			
ADD r	1 0 0 0 0 S S S	Add register to A	4
ADC r	1 0 0 0 1 S S S	Add register to A with carry	4
ADD M	1 0 0 0 0 1 1 0	Add memory to A	7
ADC M	1 0 0 0 1 1 1 0	Add memory to A with carry	7
ADI	1 1 0 0 0 1 1 0	Add immediate to A	7
ACI	1 1 0 0 1 1 1 0	Add immediate to A with carry	7
DAD B	0 0 0 0 1 0 0 1	Add B & C to H & L	10
DAD D	0 0 0 1 1 0 0 1	Add D & E to H & L	10
DAD H	0 0 1 0 1 0 0 1	Add H & L to H & L	10
DAD SP	0 0 1 1 1 0 0 1	Add stack pointer to H & L	10

Table 2. Instruction Set Summary (Continued)

Mnemonic*	Instruction Code (1) D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0	Operations Description	Clock Cycles (2)
SUBTRACT			
SUB r	1 0 0 1 0 S S S	Subtract register from A	4
SBB r	1 0 0 1 1 S S S	Subtract register from A with borrow	4
SUB M	1 0 0 1 0 1 1 0	Subtract memory from A	7
SBB M	1 0 0 1 1 1 1 0	Subtract memory from A with borrow	7
SUI	1 1 0 1 0 1 1 0	Subtract immediate from A	7
SBI	1 1 0 1 1 1 1 0	Subtract immediate from A with borrow	7
LOGICAL			
ANA r	1 0 1 0 0 S S S	And register with A	4
XRA r	1 0 1 0 1 S S S	Exclusive or register with A	4
ORA r	1 0 1 1 0 S S S	Or register with A	4
CMP r	1 0 1 1 1 S S S	Compare register with A	4
ANA M	1 0 1 0 0 1 1 0	And memory with A	7
XRA M	1 0 1 0 1 1 1 0	Exclusive Or memory with A	7
ORA M	1 0 1 1 0 1 1 0	Or memory with A	7
CMP M	1 0 1 1 1 1 1 0	Compare memory with A	7
ANI	1 1 1 0 0 1 1 0	And immediate with A	7
XRI	1 1 1 0 1 1 1 0	Exclusive Or immediate with A	7
ORI	1 1 1 1 0 1 1 0	Or immediate with A	7
CPI	1 1 1 1 1 1 1 0	Compare immediate with A	7

Mnemonic*	Instruction Code (1) D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0	Operations Description	Clock Cycles (2)
ROTATE			
RLC	0 0 0 0 0 1 1 1	Rotate A left	4
RRC	0 0 0 0 1 1 1 1	Rotate A right	4
RAL	0 0 0 1 0 1 1 1	Rotate A left through carry	4
RAR	0 0 0 1 1 1 1 1	Rotate A right through carry	4
SPECIALS			
CMA	0 0 1 0 1 1 1 1	Complement A	4
STC	0 0 1 1 0 1 1 1	Set carry	4
CMC	0 0 1 1 1 1 1 1	Complement carry	4
DAA	0 0 1 0 0 1 1 1	Decimal adjust A	4
INPUT/OUTPUT			
IN	1 1 0 1 1 0 1 1	Input	10
OUT	1 1 0 1 0 0 1 1	Output	10
CONTROL			
EI	1 1 1 1 1 0 1 1	Enable Interrupts	4
DI	1 1 1 1 0 0 1 1	Disable Interrupt	4
NOP	0 0 0 0 0 0 0 0	No-operation	4
HLT	0 1 1 1 0 1 1 0	Halt	7

NOTES:
1. DDD or SSS: B = 000, C = 001, D = 010, E = 011, H = 100, L = 101, Memory = 110, A = 111.
2. Two possible cycle times (6/12) indicate instruction cycles dependent on condition flags.
*All mnemonics copyright © Intel Corporation 1977

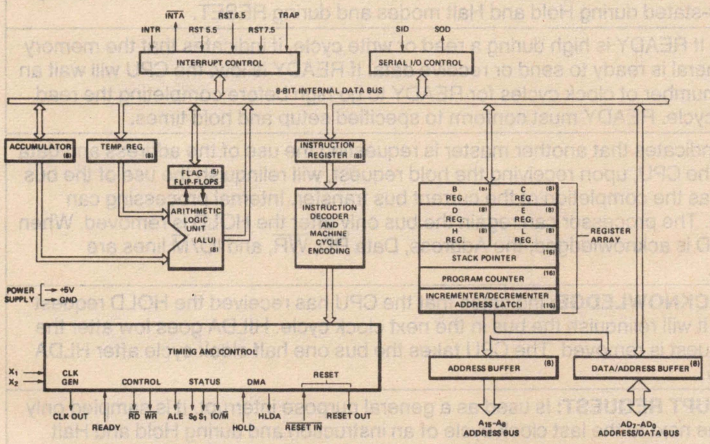
8085AH/8085AH-2/8085AH-1
8-BIT HMOS MICROPROCESSORS

- **Single +5V Power Supply with 10% Voltage Margins**
- **3 MHz, 5 MHz and 6 MHz Selections Available**
- **20% Lower Power Consumption than 8085A for 3 MHz and 5 MHz**
- **1.3 μs Instruction Cycle (8085AH); 0.8 μs (8085AH-2); 0.67 μs (8085AH-1)**
- **100% Software Compatible with 8080A**
- **On-Chip Clock Generator (with External Crystal, LC or RC Network)**
- **On-Chip System Controller; Advanced Cycle Status Information Available for Large System Control**
- **Four Vectored Interrupt Inputs (One Is Non-Maskable) Plus an 8080A-Compatible Interrupt**
- **Serial In/Serial Out Port**
- **Decimal, Binary and Double Precision Arithmetic**
- **Direct Addressing Capability to 64K Bytes of Memory**
- **Available in 40-Lead Cerdip and Plastic Packages**
  (See Packaging Spec., Order #231369)

The Intel 8085AH is a complete 8-bit parallel Central Processing Unit (CPU) implemented in N-channel, depletion load, silicon gate technology (HMOS). Its instruction set is 100% software compatible with the 8080A microprocessor, and it is designed to improve the present 8080A's performance by higher system speed. Its high level of system integration allows a minimum system of three IC's [8085AH (CPU), 8156H (RAM/IO) and 8755A (EPROM/IO)] while maintaining total system expandability. The 8085AH-2 and 8085AH-1 are faster versions of the 8085AH.

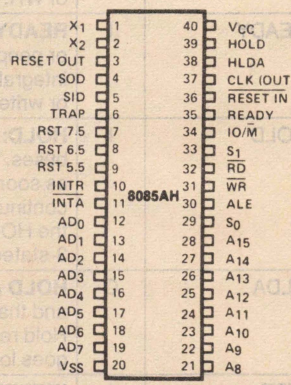
The 8085AH incorporates all of the features that the 8224 (clock generator) and 8228 (system controller) provided for the 8080A, thereby offering a higher level of system integration.

The 8085AH uses a multiplexed data bus. The address is split between the 8-bit address bus and the 8-bit data bus. The on-chip address latches of 8155H/8156H/8755A memory products allow a direct interface with the 8085AH.



231718-1

Figure 1. 8085AH CPU Functional Block Diagram



231718-2

Figure 2. 8085AH Pin Configuration

November 1986
Order Number: 231718-001

Table 1. Pin Description

Symbol	Type	Name and Function
A_8–A_{15}	O	**ADDRESS BUS:** The most significant 8 bits of memory address or the 8 bits of the I/O address, 3-stated during Hold and Halt modes and during RESET.
AD_{0-7}	I/O	**MULTIPLEXED ADDRESS/DATA BUS:** Lower 8 bits of the memory address (or I/O address) appear on the bus during the first clock cycle (T state) of a machine cycle. It then becomes the data bus during the second and third clock cycles.
ALE	O	**ADDRESS LATCH ENABLE:** It occurs during the first clock state of a machine cycle and enables the address to get latched into the on-chip latch of peripherals. The falling edge of ALE is set to guarantee setup and hold times for the address information. The falling edge of ALE can also be used to strobe the status information. ALE is never 3-stated.
S_0, S_1 and IO/$\overline{M}$	O	**MACHINE CYCLE STATUS:** **IO/$\overline{M}$ S_1 S_0 Status** 0 0 1 Memory write 0 1 0 Memory read 1 0 1 I/O write 1 1 0 I/O read 0 1 1 Opcode fetch 1 1 1 Interrupt Acknowledge * 0 0 Halt * X X Hold * X X Reset * = 3-state (high impedance) X = unspecified S_1 can be used as an advanced R/$\overline{W}$ status. IO/$\overline{M}$, S_0 and S_1 become valid at the beginning of a machine cycle and remain stable throughout the cycle. The falling edge of ALE may be used to latch the state of these lines.
$\overline{RD}$	O	**READ CONTROL:** A low level on $\overline{RD}$ indicates the selected memory or I/O device is to be read and that the Data Bus is available for the data transfer, 3-stated during Hold and Halt modes and during RESET.
$\overline{WR}$	O	**WROTE CONTROL:** A low level on $\overline{WR}$ indicates the data on the Data Bus is to be written into the selected memory or I/O location. Data is set up at the trailing edge of $\overline{WR}$. 3-stated during Hold and Halt modes and during RESET.
READY	I	**READY:** If READY is high during a read or write cycle, it indicates that the memory or peripheral is ready to send or receive data. If READY is low, the CPU will wait an integral number of clock cycles for READY to go high before completing the read or write cycle. READY must conform to specified setup and hold times.
HOLD	I	**HOLD:** Indicates that another master is requesting the use of the address and data buses. The CPU, upon receiving the hold request, will relinquish the use of the bus as soon as the completion of the current bus transfer. Internal processing can continue. The processor can regain the bus only after the HOLD is removed. When the HOLD is acknowledged, the Address, Data $\overline{RD}$, $\overline{WR}$, and IO/$\overline{M}$ lines are 3-stated.
HLDA	O	**HOLD ACKNOWLEDGE:** Indicates that the CPU has received the HOLD request and that it will relinquish the bus in the next clock cycle. HILDA goes low after the Hold request is removed. The CPU takes the bus one half clock cycle after HLDA goes low.
INTR	I	**INTERRUPT REQUEST:** Is used as a general purpose interrupt. It is sampled only during the next to the last clock cycle of an instruction and during Hold and Halt states. If it is active, the Program Counter (PC) will be inhibited from incrementing and an $\overline{INTA}$ will be issued. During this cycle a RESTART or CALL instruction can be inserted to jump to the interrupt service routine. The INTR is enabled and disabled by software. It is disabled by Reset and immediately after an interrupt is accepted.

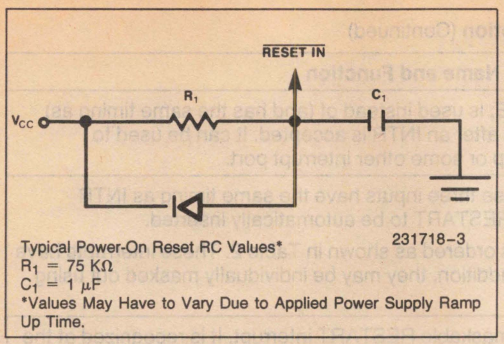
Table 1. Pin Description (Continued)

Symbol	Type	Name and Function
$\overline{INTA}$	O	**INTERRUPT ACKNOWLEDGE:** Is used instead of (and has the same timing as) $\overline{RD}$ during the Instruction cycle after an INTR is accepted. It can be used to activate an 8259A Interrupt chip or some other interrupt port.
RST 5.5 RST 6.5 RST 7.5	I	**RESTART INTERRUPTS:** These three inputs have the same timing as INTR except they cause an internal RESTART to be automatically inserted. The priority of these interrupt is ordered as shown in Table 2. These interrupts have a higher priority than INTR. In addition, they may be individually masked out using the SIM instruction.
TRAP	I	**TRAP:** Trap interrupt is a non-maskable RESTART interrupt. It is recognized at the same time as INTR or RST 5.5–7.5. It is unaffected by any mask or Interrupt Enable. It has the highest priority of any interrupt. (See Table 2.)
$\overline{RESET\ IN}$	I	**RESET IN:** Sets the Program Counter to zero and resets the Interrupt Enable and HLDA flip-flops. The data and address buses and the control lines are 3-stated during RESET and because of the asynchronous nature of RESET, the processor's internal registers and flags may be altered by RESET with unpredictable results. $\overline{RESET\ IN}$ is a Schmitt-triggered input, allowing connection to an R-C network for power-on RESET delay (see Figure 3). Upon power-up, $\overline{RESET\ IN}$ must remain low for at least 10 ms after minimum V_{CC} has been reached. For proper reset operation after the power-up duration, $\overline{RESET\ IN}$ should be kept low a minimum of three clock periods. The CPU is held in the reset condition as long as $\overline{RESET\ IN}$ is applied.
RESET OUT	O	**RESET OUT:** Reset Out indicates CPU is being reset. Can be used as a system reset. The signal is synchronized to the processor clock and lasts an integral number of clock periods.
X_1, X_2	I	**X_1 and X_2:** Are connected to a crystal, LC, or RC network to drive the internal clock generator. X_1 can also be an external clock input from a logic gate. The input frequency is divided by 2 to give the processor's internal operating frequency.
CLK	O	**CLOCK:** Clock output for use as a system clock. The period of CLK is twice the X_1, X_2 input period.
SID	I	**SERIAL INPUT DATA LINE:** The data on this line is loaded into accumulator bit 7 whenever a RIM instruction is executed.
SOD	O	**SERIAL OUTPUT DATA LINE:** The output SOD is set or reset as specified by the SIM instruction.
V_{CC}		**POWER:** +5 volt supply.
V_{SS}		**GROUND:** Reference.

Table 2. Interrupt Priority, Restart Address and Sensitivity

Name	Priority	Address Branched to(1) When Interrupt Occurs	Type Trigger
TRAP	1	24H	Rising Edge AND High Level until Sampled
RST 7.5	2	3CH	Rising Edge (Latched)
RST 6.5	3	34H	High Level until Sampled
RST 5.5	4	2CH	High Level until Sampled
INTR	5	(Note 2)	High Level until Sampled

NOTES:
1. The processor pushes the PC on the stack before branching to the indicated address.
2. The address branched to depends on the instruction provided to the CPU when the interrupt is acknowledged.



Typical Power-On Reset RC Values*
R_1 = 75 KΩ
C_1 = 1 μF
*Values May Have to Vary Due to Applied Power Supply Ramp Up Time.

Figure 3. Power-On Reset Circuit

FUNCTIONAL DESCRIPTION

The 8085AH is a complete 8-bit parallel central processor. It is designed with N-channel, depletion load, silicon gate technology (HMOS), and requires a single +5V supply. Its basic clock speed is 3 MHz (8085AH), 5 MHz (8085AH-2), or 6 MHz (8085-AH-1), thus improving on the present 8080A's performance with higher system speed. Also it is designed to fit into a minimum system of three IC's: The CPU (8085AH), a RAM/IO (8156H), and an EPROM/IO chip (8755A).

The 8085AH has twelve addressable 8-bit registers. Four of them can function only as two 16-bit register pairs. Six others can be used interchangeably as 8-bit registers or as 16-bit register pairs. The 8085AH register set is as follows:

Mnemonic	Register	Contents
ACC or A	Accumulator	8 Bits
PC	Program Counter	16-Bit Address
BC, DE, HL	General-Purpose Registers; data pointer (HL)	8-Bits x 6 or 16 Bits x 3
SP	Stack Pointer	16-Bit Address
Flags or F	Flag Register	5 Flags (8-Bit Space)

The 8085AH uses a multiplexed Data Bus. The address is split between the higher 8-bit Address Bus and the lower 8-bit Address/Data Bus. During the first T state (clock cycle) of a machine cycle the low order address is sent out on the Address/Data bus. These lower 8 bits may be latched externally by the Address Latch Enable signal (ALE). During the rest of the machine cycle the data bus is used for memory or I/O data.

The 8085AH provides $\overline{RD}$, $\overline{WR}$, S_0, S_1, and IO/$\overline{M}$ signals for bus control. An Interrupt Acknowledge signal ($\overline{INTA}$) is also provided. HOLD and all Interrupts are synchronized with the processor's internal clock. The 8085AH also provides Serial Input Data (SID) and Serial Output Data (SOD) lines for simple serial interface.

In addition to these features, the 8085AH has three maskable, vector interrupt pins, one nonmaskable TRAP interrupt, and a bus vectored interrupt, INTR.

INTERRUPT AND SERIAL I/O

The 8085AH has 5 interrupt inputs: INTR, RST 5.5, RST 6.5, RST 7.5, and TRAP. INTR is identical in function to the 8080A INT. Each of the three RESTART inputs, 5.5, 6.5, and 7.5, has a programmable mask. TRAP is also a RESTART interrupt but it is nonmaskable.

The three maskable interrupt cause the internal execution of RESTART (saving the program counter in the stack and branching to the RESTART address) if the interrupts are enabled and if the interrupt mask is not set. The nonmaskable TRAP causes the internal execution of a RESTART vector independent of the state of the interrupt enable or masks. (See Table 2.)

There are two different types of inputs in the restart interrupts. RST 5.5 and RST 6.5 are *high level-sensitive* like INTR (and INT on the 8080) and are recognized with the same timing as INTR. RST 7.5 is *rising edge-sensitive.*

For RST 7.5, only a pulse is required to set an internal flip-flop which generates the internal interrupt request (a normally high level signal with a low going pulse is recommended for highest system noise immunity). The RST 7.5 request flip-flop remains set until the request is serviced. Then it is reset automatically. This flip-flop may also be reset by using the SIM instruction or by issuing a $\overline{RESET\ IN}$ to the 8085AH. The RST 7.5 internal flip-flop will be set by a pulse on the RST 7.5 pin even when the RST 7.5 interrupt is masked out.

The status of the three RST interrupt masks can only be affected by the SIM instruction and $\overline{RESET\ IN}$. (See SIM, Chapter 5 of the 8080/8085 User's Manual.)

The interrupts are arranged in a fixed priority that determines which interrupt is to be recognized if more than one is pending as follows: TRAP—highest priority, RST 7.5, RST 6.5, RST 5.5, INTR—lowest priority. This priority scheme does not take into account the priority of a routine that was started by a higher priority interrupt. RST 5.5 can interrupt an RST 7.5 routine if the interrupts are re-enabled before the end of the RST 7.5 routine.

The TRAP interrupt is useful for catastrophic events such as power failure or bus error. The TRAP input is recognized just as any other interrupt but has the

highest priority. It is not affected by any flag or mask. The TRAP input is both *edge and level sensitive.* The TRAP input must go high and remain high until it is acknowledged. It will not be recognized again until it goes low, then high again. This avoids any false triggering due to noise or logic glitches. Figure 4 illustrates the TRAP interrupt request circuitry within the 8085AH. Note that the servicing of any interrupt (TRAP, RST 7.5, RST 6.5, RST 5.5, INTR) disables all future interrupts (except TRAPs) until an EI instruction is executed.

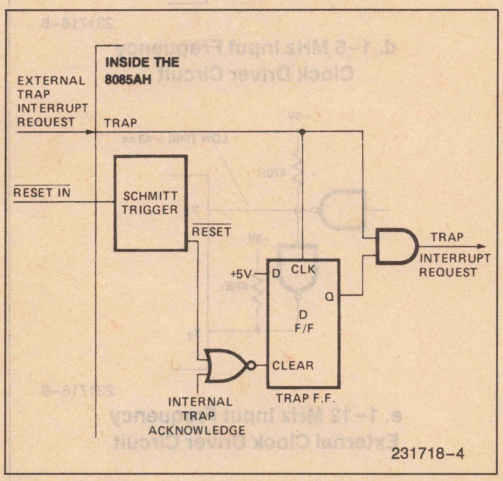


Figure 4. TRAP and RESET In Circuit

The TRAP interrupt is special in that it disables interrupts, but preserves the previous interrupt enable status. Performing the first RIM instruction following a TRAP interrupt allows you to determine whether interrupts were enabled or disabled prior to the TRAP. All subsequent RIM instructions provide current interrupt enable status. Performing a RIM instruction following INTR, or RST 5.5–7.5 will provide current Interrupt Enable status, revealing that interrupts are disabled. See the description of the RIM instruction in the 8080/8085 Family User's Manual.

The serial I/O system is also controlled by the RIM and SIM instruction. SID is read by RIM, and SIM sets the SOD data.

DRIVING THE X_1 AND X_2 INPUTS

You may drive the clock inputs of the 8085AH, 8085AH-2, or 8085AH-1 with a crystal, an LC tuned circuit, an RC network, or an external clock source. The crystal frequency must be at least 1 MHz, and must be twice the desired internal clock frequency; hence, the 8085AH is operated with a 6 MHz crystal (for 3 MHz clock), the 8085AH-2 operated with a 10 MHz crystal (for 5 MHz clock), and the 8085AH-1 can be operated with a 12 MHz crystal (for 6 MHz clock). If a crystal is used, it must have the following characteristics:

Parallel resonance at twice the clock frequency desired
C_L (load capacitance) $\leq$ 30 pF
C_S (Shunt capacitance) $\leq$ 7 pF
R_S (equivalent shunt resistance) $\leq 75\Omega$
Drive level: 10 mW
Frequency tolerance: $\pm 0.005\%$ (suggested)

Note the use of the 20 pF capacitor between X_2 and ground. This capacitor is required with crystal frequencies below 4 MHz to assure oscillator startup at the correct frequency. A parallel-resonant LC citcuit may be used as the frequency-determining network for the 8085AH, providing that its frequency tolerance of approximately $\pm 10\%$ is acceptable. The components are chosen from the formula:

$$f = \frac{1}{2\pi\sqrt{L(C_{ext} + C_{int})}}$$

To minimize variations in frequency, it is recommended that you choose a value for C_{ext} that is at least twice that of C_{int}, or 30 pF. The use of an LC circuit is not recommended for frequencies higher than approximately 5 MHz.

An RC circuit may be used as the frequency-determining network for the 8085AH if maintaining a precise clock frequency is of no importance. Variations in the on-chip timing generation can cause a wide variation in frequency when using the RC mode. Its advantage is its low component cost. The driving frequency generated by the circuit shown is approximately 3 MHz. It is not recommended that frequencies greatly higher or lower than this be attempted.

Figure 5 shows the recommended clock driver circuits. Note in d and e that pullup resistors are required to assure that the high level voltage of the input is at least 4V and maximum low level voltage of 0.8V.

For driving frequencies up to and including 6 MHz you may supply the driving signal to X_1 and leave X_2 open-circuited (Figure 5d). If the driving frequency is from 6 MHz to 12 MHz, stability of the clock generator will be improved by driving both X_1 and X_2 with a push-pull source (Figure 5e). To prevent self-oscillation of the 8085AH, be sure that X_2 is not coupled back to X_1 through the driving circuit.

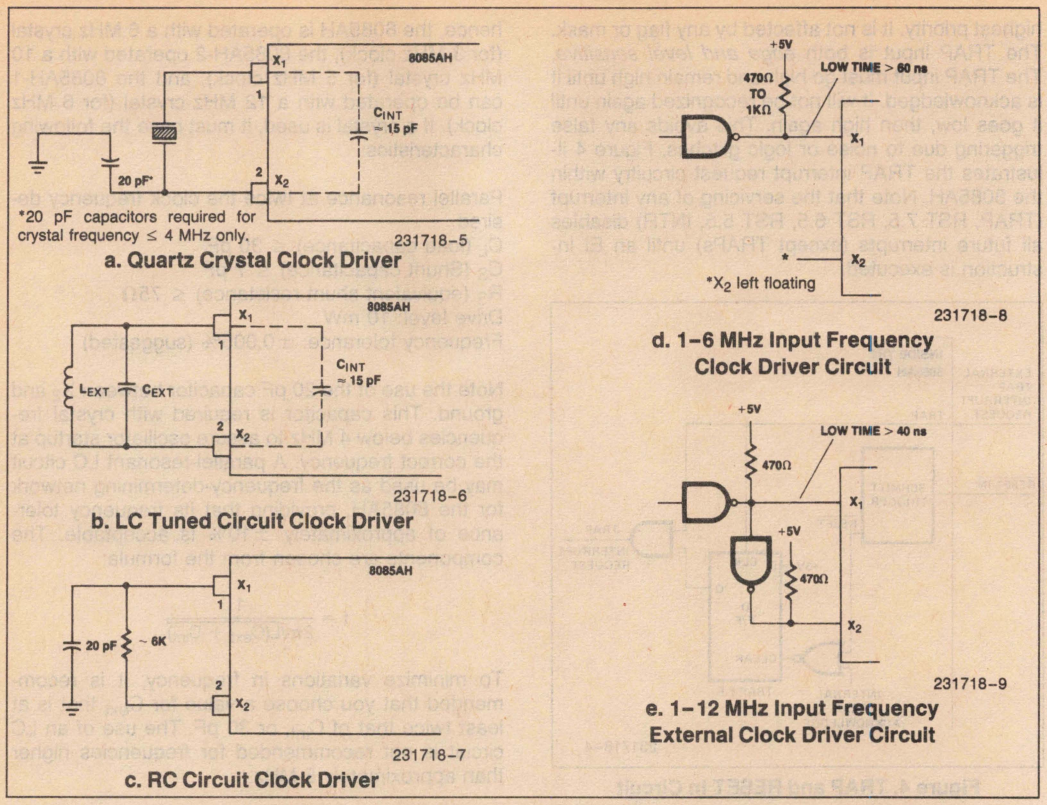


Figure 5. Clock Driver Circuits

GENERATING AN 8085AH WAIT STATE

If your system requirements are such that slow memories or peripheral devices are being used, the circuit shown in Figure 6 may be used to insert one WAIT state in each 8085AH machine cycle.

The D flip-flops should be chosen so that
- CLK is rising edge-triggered
- CLEAR is low-level active.

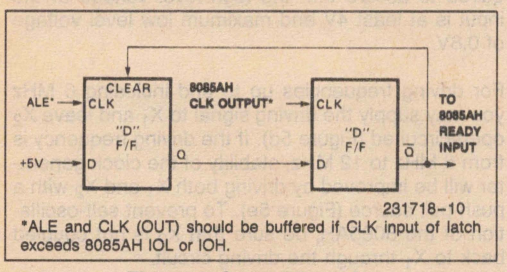


Figure 6. Generation of a Wait State for 8085AH CPU

As in the 8080, the READY line is used to extend the read and write pulse lengths so that the 8085AH can be used with slow memory. HOLD causes the CPU to relinquish the bus when it is through with it by floating the Address and Data Buses.

SYSTEM INTERFACE

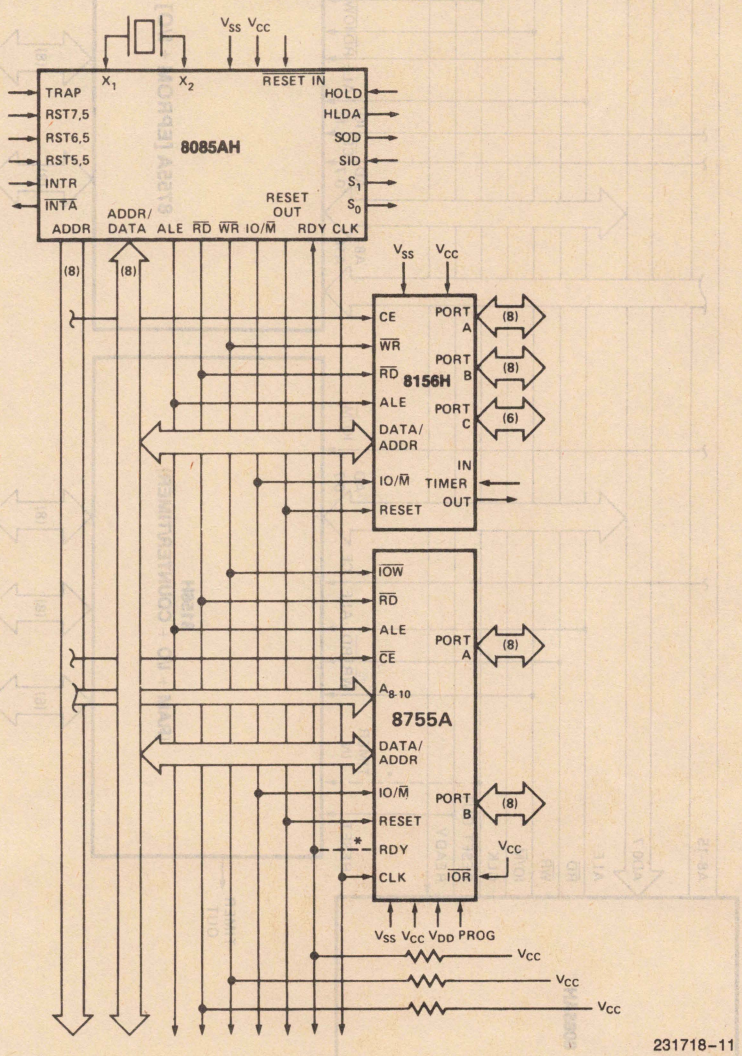
The 8085AH family includes memory components, which are directly compatible to the 8085AH CPU. For example, a system consisting of the three chips, 8085AH, 8156H and 8755A will have the following features:

- 2K Bytes EPROM
- 256 Bytes RAM
- 1 Timer/Counter
- 4 8-bit I/O Ports
- 1 6-bit I/O Port
- 4 Interrupt Levels
- Serial In/Serial Out Ports

This minimum system, using the standard I/O technique is as shown in Figure 7.

In addition to the standard I/O, the memory mapped I/O offers an efficient I/O addressing technique. With this technique, an area of memory address space is assigned for I/O address, thereby, using the memory address for I/O manipulation. Figure 8 shows the system configuration of Memory Mapped I/O using 8085AH.

The 8085AH CPU can also interface with the standard memory that does *not* have the multiplexed address/data bus. It will require a simple 8-bit latch as shown in Figure 9.



***NOTE:**
Optional Connection

Figure 7. 8085AH Minimum System (Standard I/O Technique)

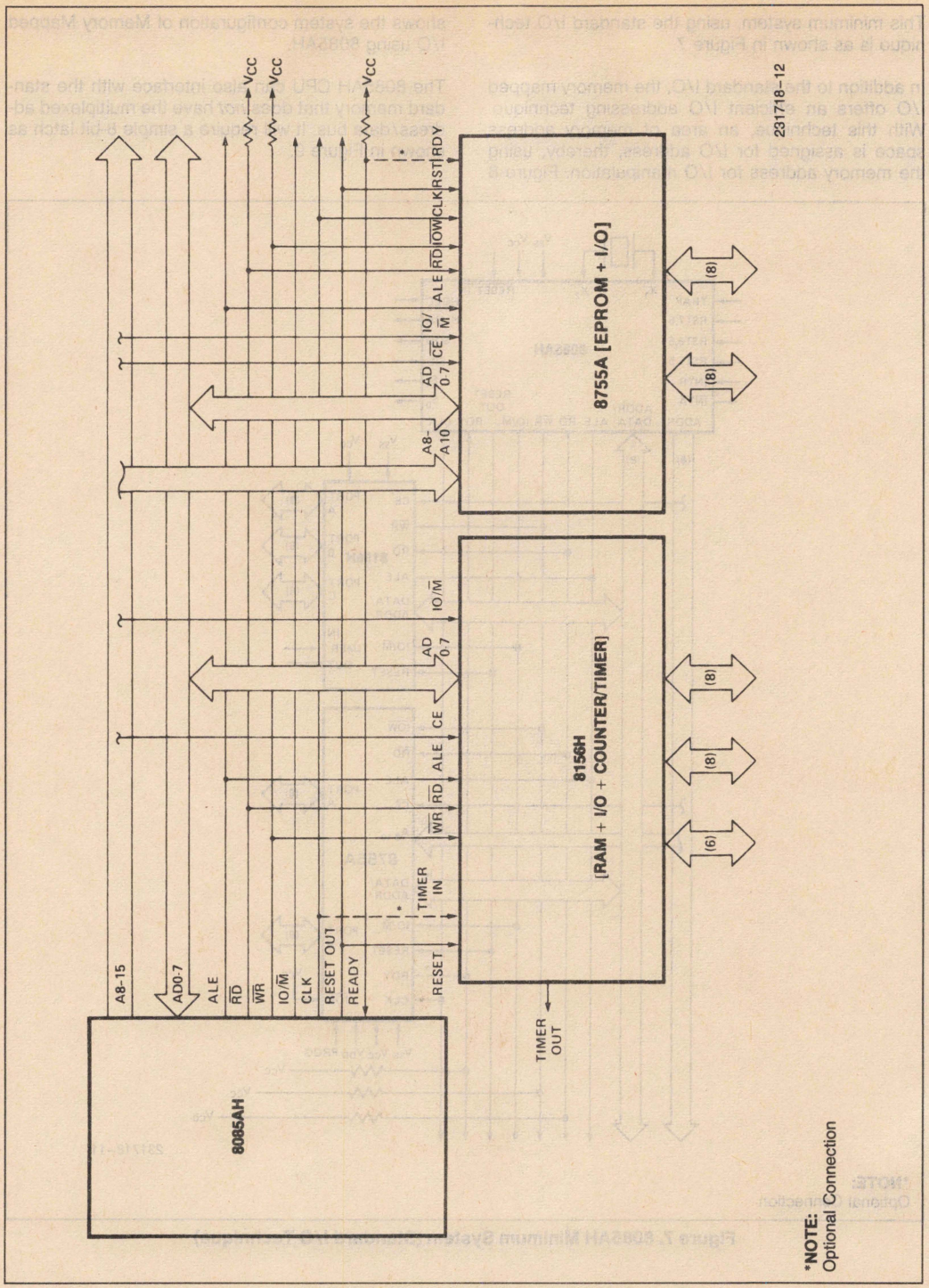


Figure 8. 8085 Minimum System (Memory Mapped I/O)

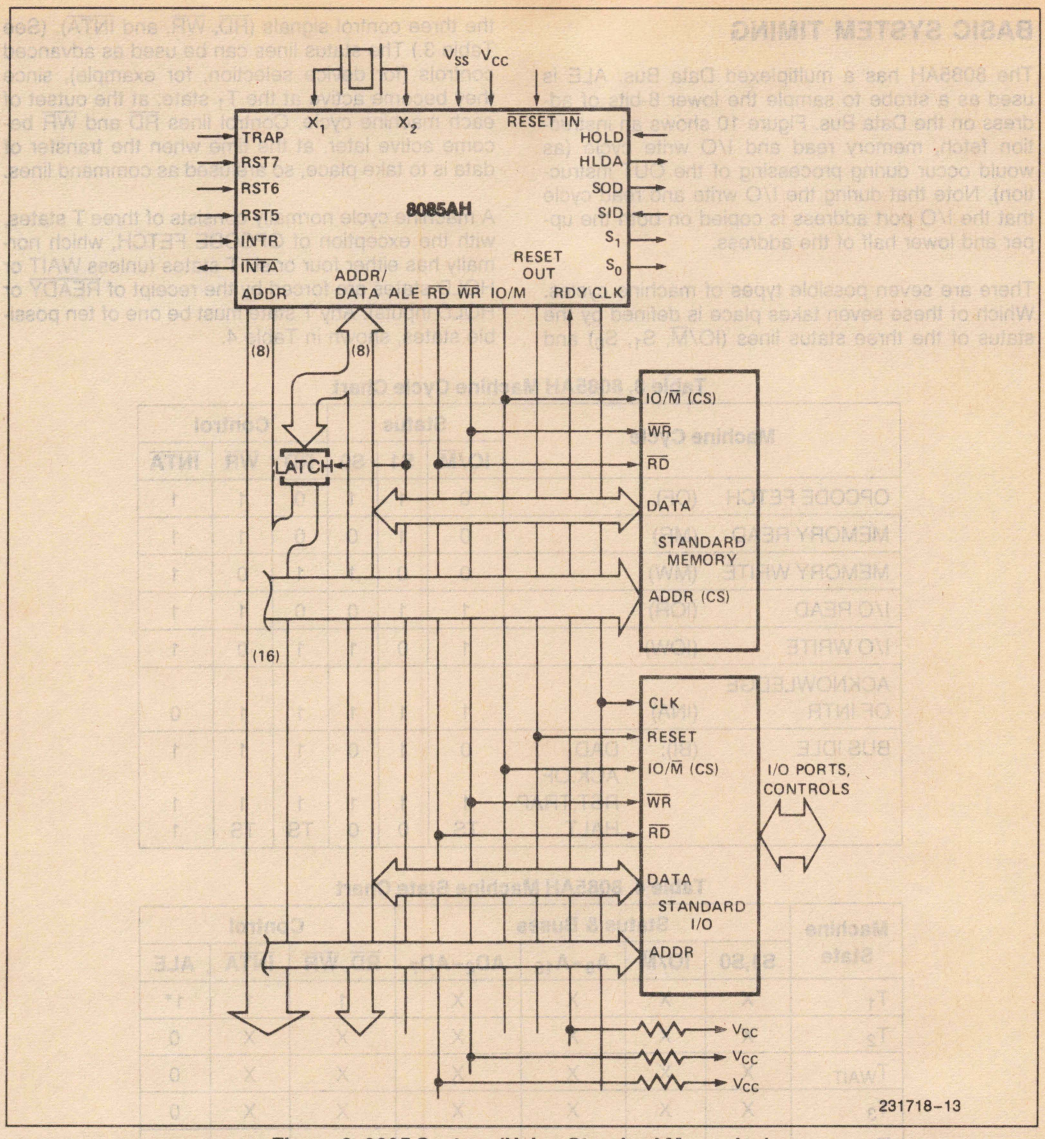


Figure 9. 8085 System (Using Standard Memories)

BASIC SYSTEM TIMING

The 8085AH has a multiplexed Data Bus. ALE is used as a strobe to sample the lower 8-bits of address on the Data Bus. Figure 10 shows an instruction fetch, memory read and I/O write cycle (as would occur during processing of the OUT instruction). Note that during the I/O write and read cycle that the I/O port address is copied on both the upper and lower half of the address.

There are seven possible types of machine cycles. Which of these seven takes place is defined by the status of the three status lines ($IO/\overline{M}$, S_1, S_0) and the three control signals ($\overline{RD}$, $\overline{WR}$, and $\overline{INTA}$). (See Table 3.) The status lines can be used as advanced controls (for device selection, for example), since they become active at the T_1 state, at the outset of each machine cycle. Control lines $\overline{RD}$ and $\overline{WR}$ become active later, at the time when the transfer of data is to take place, so are used as command lines.

A machine cycle normally consists of three T states, with the exception of OPCODE FETCH, which normally has either four or six T states (unless WAIT or HOLD states are forced by the receipt of $\overline{READY}$ or HOLD inputs). Any T state must be one of ten possible states, shown in Table 4.

Table 3. 8085AH Machine Cycle Chart

Machine Cycle			Status			Control		
			IO/$\overline{M}$	S1	S0	$\overline{RD}$	$\overline{WR}$	$\overline{INTA}$
OPCODE FETCH	(OF)		0	1	1	0	1	1
MEMORY READ	(MR)		0	1	0	0	1	1
MEMORY WRITE	(MW)		0	0	1	1	0	1
I/O READ	(IOR)		1	1	0	0	1	1
I/O WRITE	(IOW)		1	0	1	1	0	1
ACKNOWLEDGE OF INTR	(INA)		1	1	1	1	1	0
BUS IDLE	(BI):	DAD	0	1	0	1	1	1
		ACK.OF RST,TRAP	1	1	1	1	1	1
		HALT	TS	0	0	TS	TS	1

Table 4. 8085AH Machine State Chart

Machine State	Status & Buses				Control		
	S1,S0	IO/$\overline{M}$	A_8–A_{15}	AD_0–AD_7	$\overline{RD}$, $\overline{WR}$	$\overline{INTA}$	ALE
T_1	X	X	X	X	1	1	1*
T_2	X	X	X	X	X	X	0
T_{WAIT}	X	X	X	X	X	X	0
T_3	X	X	X	X	X	X	0
T_4	1	0†	X	TS	1	1	0
T_5	1	0†	X	TS	1	1	0
T_6	1	0†	X	TS	1	1	0
T_{RESET}	X	TS	TS	TS	TS	1	0
T_{HALT}	0	TS	TS	TS	TS	1	0
T_{HOLD}	X	TS	TS	TS	TS	1	0

0 = Logic "0" TS = High Impedance
1 = Logic "1" X = Unspecified
*ALE not generated during 2nd and 3rd machine cycles of DAD instruction.
†$IO/\overline{M}$ = 1 during T_4–T_6 of INA machine cycle.

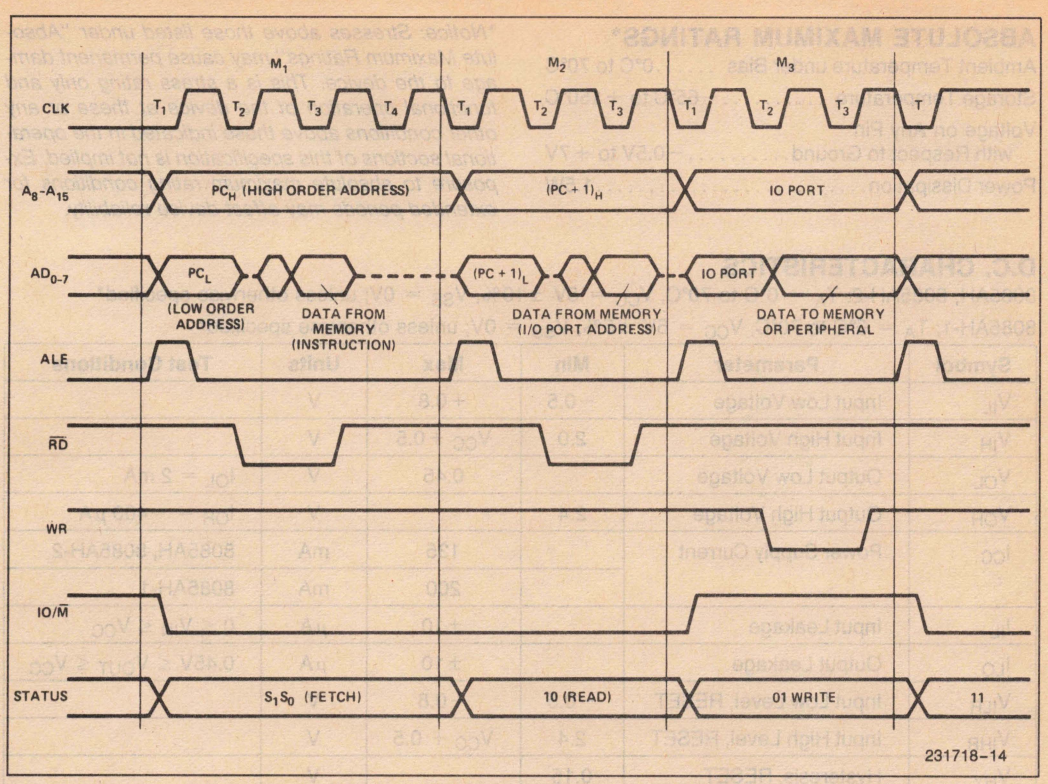


Figure 10. 8085AH Basic System Timing

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature under Bias .....0°C to 70°C
Storage Temperature .....−65°C to +150°C
Voltage on Any Pin
with Respect to Ground.....−0.5V to +7V
Power Dissipation.....1.5W

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS

8085AH, 8085AH-2: T_A = 0°C to 70°C, V_{CC} = 5V ±10%, V_{SS} = 0V; unless otherwise specified*
8085AH-1: T_A = 0°C to 70°C, V_{CC} = 5V ±5%, V_{SS} = 0V; unless otherwise specified*

Symbol	Parameter	Min	Max	Units	Test Conditions
V_{IL}	Input Low Voltage	−0.5	+0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC} +0.5	V	
V_{OL}	Output Low Voltage		0.45	V	I_{OL} = 2 mA
V_{OH}	Output High Voltage	2.4		V	I_{OH} = −400 μA
I_{CC}	Power Supply Current		135	mA	8085AH, 8085AH-2
			200	mA	8085AH-1
I_{IL}	Input Leakage		±10	μA	$0 \le V_{IN} \le V_{CC}$
I_{LO}	Output Leakage		±10	μA	$0.45V \le V_{OUT} \le V_{CC}$
V_{ILR}	Input Low Level, RESET	−0.5	+0.8	V	
V_{IHR}	Input High Level, RESET	2.4	V_{CC} + 0.5	V	
V_{HY}	Hysteresis, RESET	0.15		V	

A.C. CHARACTERISTICS

8085AH, 8085AH-2: T_A = 0°C to 70°C, V_{CC} = 5V ±10%, V_{SS} = 0V*
8085AH-1: T_A = 0°C to 70°C, V_{CC} = 5V ±5%, V_{SS} = 0V

Symbol	Parameter	8085AH (2)		8085AH-2 (2)		8085AH-1 (2)		Units
		Min	Max	Min	Max	Min	Max	
t_{CYC}	CLK Cycle Period	320	2000	200	2000	167	2000	ns
t_1	CLK Low Time (Standard CLK Loading)	80		40		20		ns
t_2	CLK High Time (Standard CLK Loading)	120		70		50		ns
t_r, t_f	CLK Rise and Fall Time		30		30		30	ns
t_{XKR}	X_1 Rising to CLK Rising	20	120	20	100	20	100	ns
t_{XKF}	X_1 Rising to CLK Falling	20	150	20	110	20	110	ns
t_{AC}	A_{8-15} Valid to Leading Edge of Control (1)	270		115		70		ns
t_{ACL}	A_{0-7} Valid to Leading Edge of Control	240		115		60		ns
t_{AD}	A_{0-15} Valid to Valid Data In		575		350		225	ns
t_{AFR}	Address Float after Leading Edge of $\overline{READ}$ ($\overline{INTA}$)		0		0		0	ns
t_{AL}	A_{8-15} Valid before Trailing Edge of ALE (1)	115		50		25		ns

***NOTE:**
For Extended Temperature EXPRESS use M8085AH Electricals Parameters.

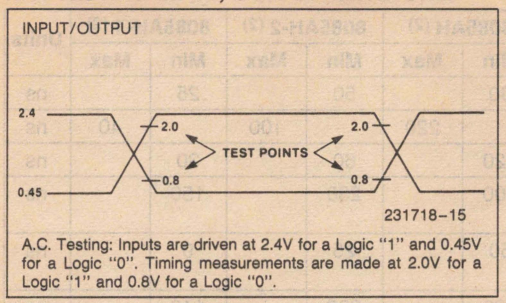
A.C. CHARACTERISTICS (Continued)

Symbol	Parameter	8085AH (2)		8085AH-2 (2)		8085AH-1 (2)		Units
		Min	Max	Min	Max	Min	Max	
t_{ALL}	A_{0-7} Valid before Trailing Edge of ALE	90		50		25		ns
t_{ARY}	READY Valid from Address Valid		220		100		40	ns
t_{CA}	Address (A_{8-15}) Valid after Control	120		60		30		ns
t_{CC}	Width of Control Low ($\overline{RD}$, $\overline{WR}$, $\overline{INTA}$) Edge of ALE	400		230		150		ns
t_{CL}	Trailing Edge of Control to Leading Edge of ALE	50		25		0		ns
t_{DW}	Data Valid to Trialing Edge of $\overline{WRITE}$	420		230		140		ns
t_{HABE}	HLDA to Bus Enable		210		150		150	ns
t_{HABF}	Bus Float after HLDA		210		150		150	ns
t_{HACK}	HLDA Valid to Trailing Edge of CLK	110		40		0		ns
t_{HDH}	HOLD Hold Time	0		0		0		ns
t_{HDS}	HOLD Setup Time to Trailing Edge of CLK	170		120		120		ns
t_{INH}	INTR Hold Time	0		0		0		ns
t_{INS}	INTR, RST, and TRAP Setup Time to Falling Edge of CLK	160		150		150		ns
t_{LA}	Address Hold Time after ALE	100		50		20		ns
t_{LC}	Trailing Edge of ALE to Leading Edge of Control	130		60		25		ns
t_{LCK}	ALE Low During CLK High	100		50		15		ns
t_{LDR}	ALE to Valid Data during Read		460		270		175	ns
t_{LDW}	ALE to Valid Data during Write		200		140		110	ns
t_{LL}	ALE Width	140		80		50		ns
t_{LRY}	ALE to READY Stable		110		30		10	ns
t_{RAE}	Trailing Edge of $\overline{READ}$ to Re-Enabling of Address	150		90		50		ns
t_{RD}	$\overline{READ}$ (or $\overline{INTA}$) to Valid Data		300		150		75	ns
t_{RV}	Control Trailing Edge to Leading Edge of Next Control	400		220		160		ns
t_{RDH}	Data Hold Time after $\overline{READ}$ $\overline{INTA}$	0		0		0		ns
t_{RYH}	READY Hold Time	0		0		5		ns
t_{RYS}	READY Setup Time to Leading Edge of CLK	110		100		100		ns
t_{WD}	Data Valid after Trailing Edge of $\overline{WRITE}$	100		60		30		ns
t_{WDL}	LEADING Edge of $\overline{WRITE}$ to Data Valid		40		20		30	ns

NOTES:

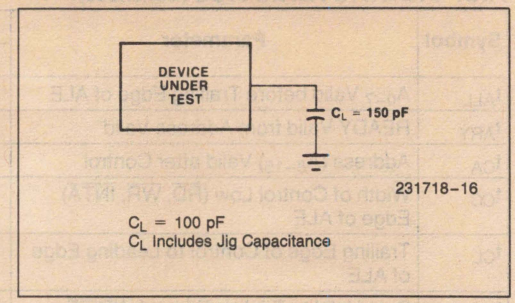
1. A_8–A_{15} address Specs apply IO/$\overline{M}$, S_0, and S_1 except A_8–A_{15} are undefined during T_4–T_6 of OF cycle whereas IO/$\overline{M}$, S_0, and S_1 are stable.
2. *Test Conditions:* t_{CYC} = 320 ns (8085AH)/200 ns (8085AH-2);/167 ns (8085AH-1); C_L = 150 pF.
3. For all output timing where C ≠ 150 pF use the following correction factors:
   25 pF ≤ C_L < 150 pF: −0.10 ns/pF
   150 pF < C_L ≤ 300 pF: +0.30 ns/pF
4. Output timings are measured with purely capacitive load.
5. To calculate timing specifications at other values of t_{CYC} use Table 5.

A.C. TESTING INPUT, OUTPUT WAVEFORM



A.C. Testing: Inputs are driven at 2.4V for a Logic "1" and 0.45V for a Logic "0". Timing measurements are made at 2.0V for a Logic "1" and 0.8V for a Logic "0".

A.C. TESTING LOAD CIRCUIT



C_L = 100 pF
C_L Includes Jig Capacitance

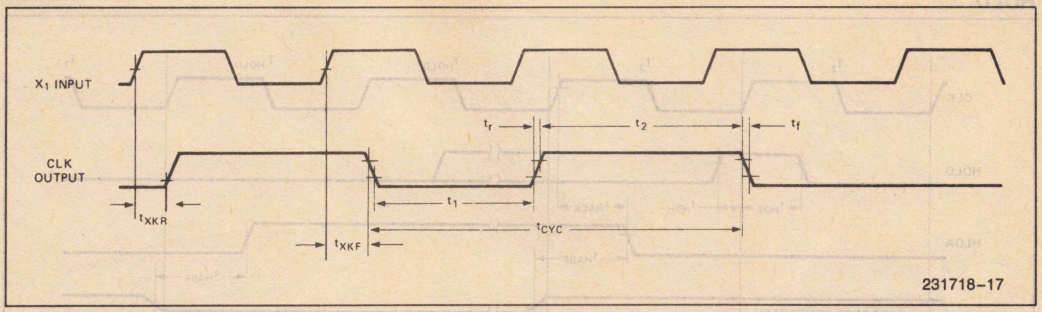
Table 5. Bus Timing Specification as a T_{CYC} Dependent

Symbol	8085AH	8085AH-2	8085AH-1	
t_{AL}	(1/2)T − 45	(1/2)T − 50	(1/2)T − 58	Minimum
t_{LA}	(1/2)T − 60	(1/2)T − 50	(1/2)T − 63	Minimum
t_{LL}	(1/2)T − 20	(1/2)T − 20	(1/2)T − 33	Minimum
t_{LCK}	(1/2)T − 60	(1/2)T − 50	(1/2)T − 68	Minimum
t_{LC}	(1/2)T − 30	(1/2)T − 40	(1/2)T − 58	Minimum
t_{AD}	(5/2 + N)T − 225	(5/2 + N)T − 150	(5/2 + N)T − 192	Maximum
t_{RD}	(3/2 + N)T − 180	(3/2 + N)T − 150	(3/2 + N)T − 175	Maximum
t_{RAE}	(1/2)T − 10	(1/2)T − 10	(1/2)T − 33	Minimum
t_{CA}	(1/2)T − 40	(1/2)T − 40	(1/2)T − 53	Minimum
t_{DW}	(3/2 + N)T − 60	(3/2 + N)T − 70	(3/2 + N)T − 110	Minimum
t_{WD}	(1/2)T − 60	(1/2)T − 40	(1/2)T − 53	Minimum
t_{CC}	(3/2 + N)T − 80	(3/2 + N)T − 70	(3/2 + N)T − 100	Minimum
t_{CL}	(1/2)T − 110	(1/2)T − 75	(1/2)T − 83	Minimum
t_{ARY}	(3/2)T − 260	(3/2)T − 200	(3/2)T − 210	Maximum
t_{HACK}	(1/2)T − 50	(1/2)T − 60	(1/2)T − 83	Minimum
t_{HABF}	(1/2)T + 50	(1/2)T + 50	(1/2)T + 67	Maximum
t_{HABE}	(1/2)T + 50	(1/2)T + 50	(1/2)T + 67	Maximum
t_{AC}	(2/2)T − 50	(2/2)T − 85	(2/2)T − 97	Minimum
t_1	(1/2)T − 80	(1/2)T − 60	(1/2)T − 63	Minimum
t_2	(1/2)T − 40	(1/2)T − 30	(1/2)T − 33	Minimum
t_{RV}	(3/2)T − 80	(3/2)T − 80	(3/2)T − 90	Minimum
t_{LDR}	(4/2)T − 180	(4/2)T − 130	(4/2)T − 159	Maximum

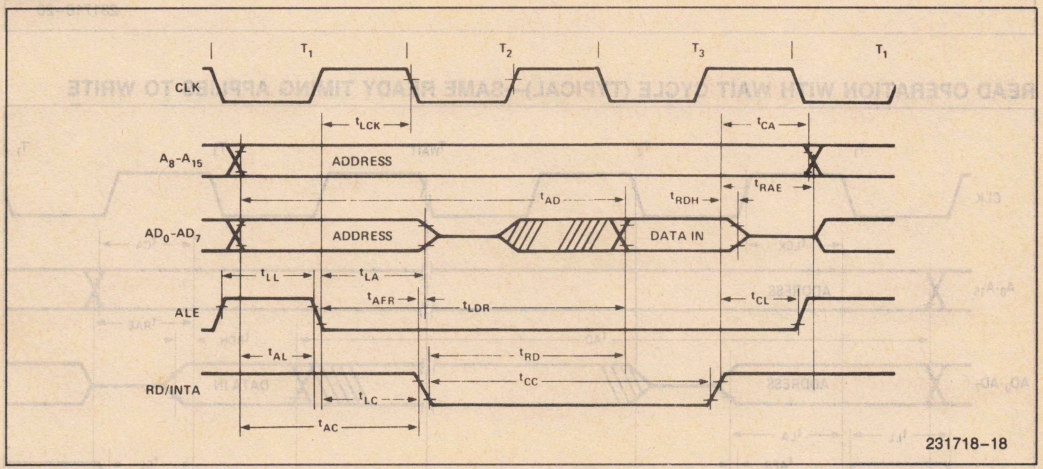
NOTE:
N is equal to the total WAIT states. T = t_{CYC}.

WAVEFORMS

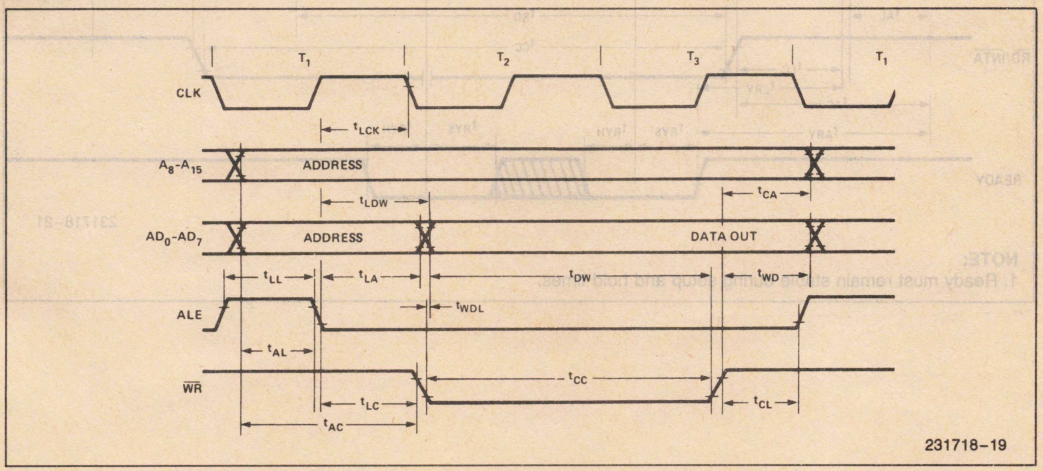
CLOCK



READ

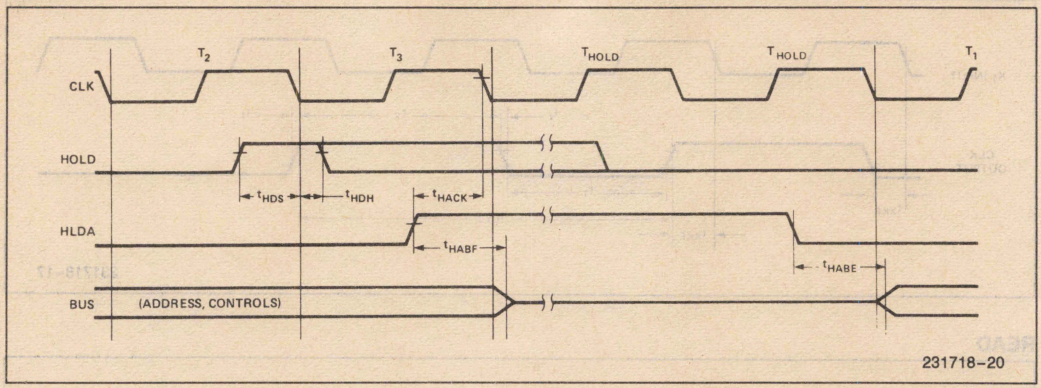


WRITE

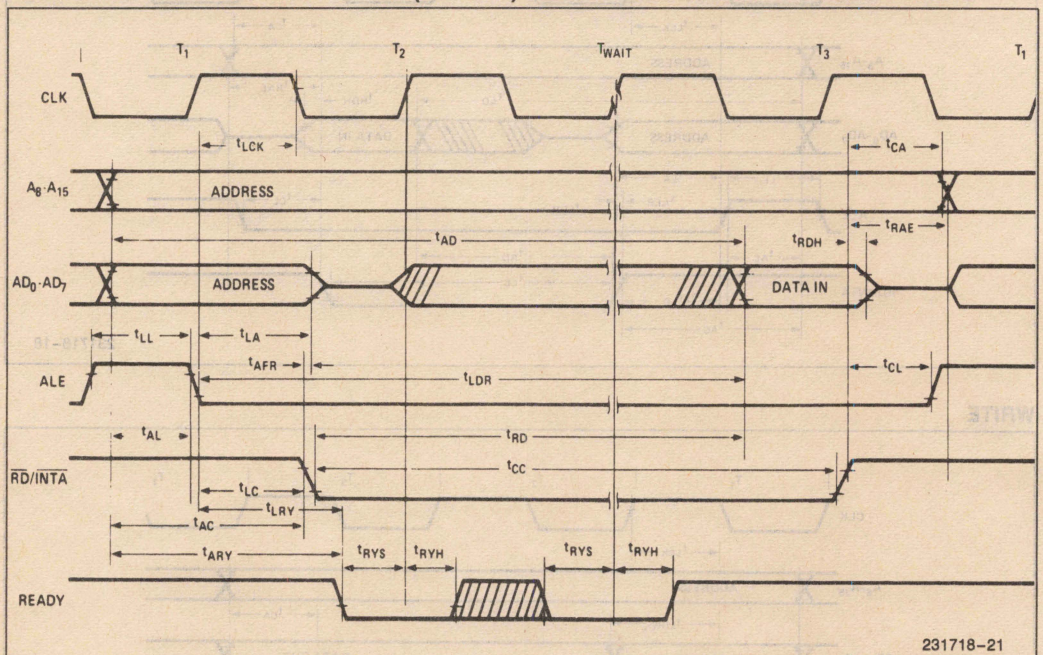


WAVEFORMS (Continued)

HOLD



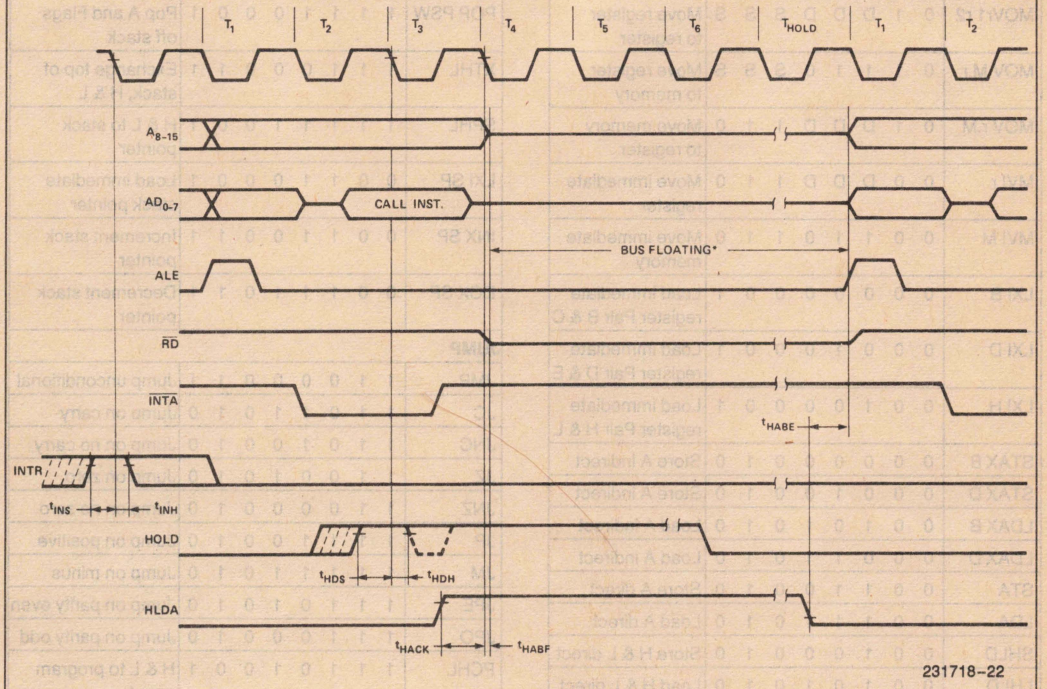
READ OPERATION WITH WAIT CYCLE (TYPICAL)—SAME READY TIMING APPLIES TO WRITE



NOTE:
1. Ready must remain stable during setup and hold times.

WAVEFORMS (Continued)

INTERRUPT AND HOLD



***NOTE:**
IO/$\overline{M}$ is also floating during this time.

Table 6. Instruction Set Summary

Mnemonic	Instruction Code D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0	Operations Description
MOVE, LOAD AND STORE		
MOVr1 r2	0 1 D D D S S S	Move register to register
MOV M.r	0 1 1 1 0 S S S	Move register to memory
MOV r.M	0 1 D D D 1 1 0	Move memory to register
MVI r	0 0 D D D 1 1 0	Move immediate register
MVI M	0 0 1 1 0 1 1 0	Move immediate memory
LXI B	0 0 0 0 0 0 0 1	Load immediate register Pair B & C
LXI D	0 0 0 1 0 0 0 1	Load immediate register Pair D & E
LXI H	0 0 1 0 0 0 0 1	Load immediate register Pair H & L
STAX B	0 0 0 0 0 0 1 0	Store A indirect
STAX D	0 0 0 1 0 0 1 0	Store A indirect
LDAX B	0 0 1 0 1 0 1 0	Load A indirect
LDAX D	0 0 0 1 1 0 1 0	Load A indirect
STA	0 0 1 1 0 0 1 0	Store A direct
LDA	0 0 1 1 1 0 1 0	Load A direct
SHLD	0 0 1 0 0 0 1 0	Store H & L direct
LHLD	0 0 1 0 1 0 1 0	Load H & L direct
XCHG	1 1 1 0 1 0 1 1	Exchange D & E, H & L Registers
STACK OPS		
PUSH B	1 1 0 0 0 1 0 1	Push register Pair B & C on stack
PUSH D	1 1 0 1 0 1 0 1	Push register Pair D & E on stack
PUSH H	1 1 1 0 0 1 0 1	Push register Pair H & L on stack
PUSH PSW	1 1 1 1 0 1 0 1	Push A and Flags on stack
POP B	1 1 0 0 0 0 0 1	Pop register Pair B & C off stack
POP D	1 1 0 1 0 0 0 1	Pop register Pair D & E off stack
POP H	1 1 1 0 0 0 0 1	Pop register Pair H & L off stack

Mnemonic	Instruction Code D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0	Operations Description
STACK OPS (Continued)		
POP PSW	1 1 1 1 0 0 0 1	Pop A and Flags off stack
XTHL	1 1 1 0 0 0 1 1	Exchange top of stack, H & L
SPHL	1 1 1 1 1 0 0 1	H & L to stack pointer
LXI SP	0 0 1 1 0 0 0 1	Load immediate stack pointer
INX SP	0 0 1 1 0 0 1 1	Increment stack pointer
DCX SP	0 0 1 1 1 0 1 1	Decrement stack pointer
JUMP		
JMP	1 1 0 0 0 0 1 1	Jump unconditional
JC	1 1 0 1 1 0 1 0	Jump on carry
JNC	1 1 0 1 0 0 1 0	Jump on no carry
JZ	1 1 0 0 1 0 1 0	Jump on zero
JNZ	1 1 0 0 0 0 1 0	Jump on no zero
JP	1 1 1 1 0 0 1 0	Jump on positive
JM	1 1 1 1 1 0 1 0	Jump on minus
JPE	1 1 1 0 1 0 1 0	Jump on parity even
JPO	1 1 1 0 0 0 1 0	Jump on parity odd
PCHL	1 1 1 0 1 0 0 1	H & L to program counter
CALL		
CALL	1 1 0 0 1 1 0 1	Call unconditional
CC	1 1 0 1 1 1 0 0	Call on carry
CNC	1 1 0 1 0 1 0 0	Call on no carry
CZ	1 1 0 0 1 1 0 0	Call on zero
CNZ	1 1 0 0 0 1 0 0	Call on no zero
CP	1 1 1 1 0 1 0 0	Call on positive
CM	1 1 1 1 1 1 0 0	Call on minus
CPE	1 1 1 0 1 1 0 0	Call on parity even
CPO	1 1 1 0 0 1 0 0	Call on parity odd
RETURN		
RET	1 1 0 0 1 0 0 1	Return
RC	1 1 0 1 1 0 0 0	Return on carry
RNC	1 1 0 1 0 0 0 0	Return on no carry
RZ	1 1 0 0 1 0 0 0	Return on zero

Table 6. Instruction Set Summary (Continued)

Mnemonic	Instruction Code D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0	Operations Description
RETURN (Continued)		
RNZ	1 1 0 0 0 0 0 0	Return on no zero
RP	1 1 1 1 0 0 0 0	Return on positive
RM	1 1 1 1 1 0 0 0	Return on minus
RPE	1 1 1 0 1 0 0 0	Return on parity even
RPO	1 1 1 0 0 0 0 0	Return on parity odd
RESTART		
RST	1 1 A A A 1 1 1	Restart
INPUT/OUTPUT		
IN	1 1 0 1 1 0 1 1	Input
OUT	1 1 0 1 0 0 1 1	Output
INCREMENT AND DECREMENT		
INR r	0 0 D D D 1 0 0	Increment register
DCR r	0 0 D D D 1 0 1	Decrement register
INR M	0 0 1 1 0 1 0 0	Increment memory
DCR M	0 0 1 1 0 1 0 1	Decrement memory
INX B	0 0 0 0 0 0 1 1	Increment B & C registers
INX D	0 0 0 1 0 0 1 1	Increment D & E registers
INX H	0 0 1 0 0 0 1 1	Increment H & L registers
DCX B	0 0 0 0 1 0 1 1	Decrement B & C
DCX D	0 0 0 1 1 0 1 1	Decrement D & E
DCX H	0 0 1 0 1 0 1 1	Decrement H & L
ADD		
ADD r	1 0 0 0 0 S S S	Add register to A
ADC r	1 0 0 0 1 S S S	Add register to A with carry
ADD M	1 0 C 0 0 1 1 0	Add memory to A
ADC M	1 0 0 0 1 1 1 0	Add memory to A with carry
ADI	1 1 0 0 0 1 1 0	Add immediate to A
ACI	1 1 0 0 1 1 1 0	Add immediate to A with carry
DAD B	0 0 0 0 1 0 0 1	Add B & C to H & L

Mnemonic	Instruction Code D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0	Operations Description
ADD (Continued)		
DAD D	0 0 0 1 1 0 0 1	Add D & E to H & L
DAD H	0 0 1 0 1 0 0 1	Add H & L to H & L
DAD SP	0 0 1 1 1 0 0 1	Add stack pointer to H & L
SUBTRACT		
SUB r	1 0 0 1 0 S S S	Subtract register from A
SBB r	1 0 0 1 1 S S S	Subtract register from A with borrow
SUB M	1 0 0 1 0 1 1 0	Subtract memory from A
SBB M	1 0 0 1 1 1 1 0	Subtract memory from A with borrow
SUI	1 1 0 1 0 1 1 0	Subtract immediate from A
SBI	1 1 0 1 1 1 1 0	Subtract immediate from A with borrow
LOGICAL		
ANA r	1 0 1 0 0 S S S	And register with A
XRA r	1 0 1 0 1 S S S	Exclusive OR register with A
ORA r	1 0 1 1 0 S S S	OR register with A
CMP r	1 0 1 1 1 S S S	Compare register with A
ANA M	1 0 1 0 0 1 1 0	And memory with A
XRA M	1 0 1 0 1 1 1 0	Exclusive OR memory with A
ORA M	1 0 1 1 0 1 1 0	OR memory with A
CMP M	1 0 1 1 1 1 1 0	Compare memory with A
ANI	1 1 1 0 0 1 1 0	And immediate with A
XRI	1 1 1 0 1 1 1 0	Exclusive OR immediate with A
ORI	1 1 1 1 0 1 1 0	OR immediate with A
CPI	1 1 1 1 1 1 1 0	Compare immediate with A

Table 6. Instruction Set Summary (Continued)

Mnemonic	Instruction Code D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0	Operations Description
ROTATE		
RLC	0 0 0 0 0 1 1 1	Rotate A left
RRC	0 0 0 0 1 1 1 1	Rotate A right
RAL	0 0 0 1 0 1 1 1	Rotate A left through carry
RAR	0 0 0 1 1 1 1 1	Rotate A right through carry
SPECIALS		
CMA	0 0 1 0 1 1 1 1	Complement A
STC	0 0 1 1 0 1 1 1	Set carry
CMC	0 0 1 1 1 1 1 1	Complement carry
DAA	0 0 1 0 0 1 1 1	Decimal adjust A

Mnemonic	Instruction Code D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0	Operations Description
CONTROL		
EI	1 1 1 1 1 0 1 1	Enable Interrupts
DI	1 1 1 1 0 0 1 1	Disable Interrupt
NOP	0 0 0 0 0 0 0 0	No-operation
HLT	0 1 1 1 0 1 1 0	Halt
NEW 8085AH INSTRUCTIONS		
RIM	0 0 1 0 0 0 0 0	Read Interrupt Mask
SIM	0 0 1 1 0 0 0 0	Set Interrupt Mask

NOTES:
1. DDS or SSS: B 000, C 001, D 010, E011, H 100, L101, Memory 110, A 111.
2. Two possible cycle times (6/12) indicate instruction cycles dependent on condition flags.

*All mnemonics copyrighted © Intel Corporation 1976.

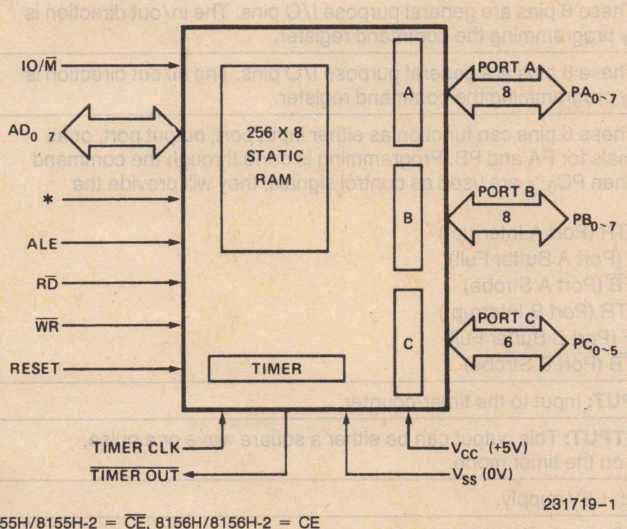
8155H/8156H/8155H-2/8156H-2
2048-BIT STATIC HMOS RAM WITH I/O PORTS AND TIMER

- **Single +5V Power Supply with 10% Voltage Margins**
- **30% Lower Power Consumption than the 8155 and 8156**
- **256 Word x 8 Bits**
- **Completely Static Operation**
- **Internal Address Latch**
- **2 Programmable 8-Bit I/O Ports**
- **1 Programmable 6-Bit I/O Port**
- **Programmable 14-Bit Binary Counter/ Timer**
- **Compatible with 8085AH and 8088 CPU**
- **Multiplexed Address and Data Bus**
- **Available in EXPRESS**
  - **— Standard Temperature Range**
  - **— Extended Temperature Range**

The Intel® 8155H and 8156H are RAM and I/O chips implemented in N-Channel, depletion load, silicon gate technology (HMOS), to be used in the 8085AH and 8088 microprocessor systems. The RAM portion is designed with 2048 static cells organized as 256 x 8. They have a maximum access time of 400 ns to permit use with no wait states in 8085AH CPU. The 8155H-2 and 8156H-2 have maximum access times of 330 ns for use with the 8085H-2 and the 5 MHz 8088 CPU.

The I/O portion consists of three general purpose I/O ports. One of the three ports can be programmed to be status pins, thus allowing the other two ports to operate in handshake mode.

A 14-bit programmable counter/timer is also included on chip to provide either a square wave or terminal count pulse for the CPU system depending on timer mode.



*8155H/8155H-2 = $\overline{CE}$, 8156H/8156H-2 = CE

Figure 1. Block Diagram

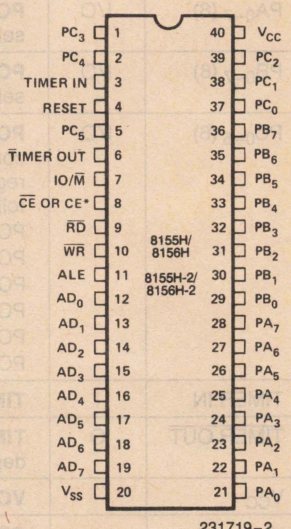


Figure 2. Pin Configuration

November 1986
Order Number: 231719-001

Table 1. Pin Description

Symbol	Type	Name and Function
RESET	I	**RESET:** Pulse provided by the 8085AH to initialize the system (connect to 8085AH RESET OUT). Input high on this line resets the chip and initializes the three I/O ports to input mode. The width of RESET pulse should typically be two 8085AH clock cycle times.
AD_{0-7}	I/O	**ADDRESS/DATA:** 3-state Address/Data lines that interface with the CPU lower 8-bit Address/Data Bus. The 8-bit address is latched into the address latch inside the 8155H/56H on the falling edge of ALE. The address can be either for the memory section or the I/O section depending on the IO/$\overline{M}$ input. The 8-bit data is either written into the chip or read from the chip, depending on the $\overline{WR}$ or $\overline{RD}$ input signal.
CE or $\overline{CE}$	I	**CHIP ENABLE:** On the 8155H, this pin is $\overline{CE}$ and is ACTIVE LOW. On the 8156H, this pin is CE and is ACTIVE HIGH.
$\overline{RD}$	I	**READ CONTROL:** Input low on this line with the Chip Enable active enables and AD_{0-7} buffers. If IO/$\overline{M}$ pin is low, the RAM content will be read out to the AD bus. Otherwise the content of the selected I/O port or command/status registers will be read to the AD bus.
$\overline{WR}$	I	**WRITE CONTROL:** Input low on this line with the Chip Enable active causes the data on the Address/Data bus to be written to the RAM or I/O ports and command/status register, depending on IO/$\overline{M}$.
ALE	I	**ADDRESS LATCH ENABLE:** This control signal latches both the address on the AD_{0-7} lines and the state of the Chip Enable and IO/$\overline{M}$ into the chip at the falling edge of ALE.
IO/$\overline{M}$	I	**I/O MEMORY:** Selects memory if low and I/O and command/status registers if high.
PA_{0-7} (8)	I/O	**PORT A:** These 8 pins are general purpose I/O pins. The in/out direction is selected by programming the command register.
PB_{0-7} (8)	I/O	**PORT B:** These 8 pins are general purpose I/O pins. The in/out direction is selected by programming the command register.
PC_{0-5} (6)	I/O	**PORT C:** These 6 pins can function as either input port, output port, or as control signals for PA and PB. Programming is done through the command register. When PC_{0-5} are used as control signals, they will provide the following: PC_0—A INTR (Port A Interrupt) PC_1—ABF (Port A Buffer Full) PC_2—$\overline{A\ STB}$ (Port A Strobe) PC_3—B INTR (Port B Interrupt) PC_4—B BF (Port B Buffer Full) PC_5—$\overline{B\ STB}$ (Port B Strobe)
TIMER IN	I	**TIMER INPUT:** Input to the timer-counter.
$\overline{TIMER\ OUT}$	O	**TIMER OUTPUT:** This output can be either a square wave or a pulse, depending on the timer mode.
V_{CC}		**VOLTAGE:** +5V supply.
V_{SS}		**GROUND:** Ground reference.

FUNCTIONAL DESCRIPTION

The 8155H/8156H contains the following:

- 2K Bit Static RAM organized as 256 x 8
- Two 8-bit I/O ports (PA & PB) and one 6-bit I/O port (PC)
- 14-bit timer-counter

The IO/$\overline{M}$ (IO/Memory Select) pin selects either the five registers (Command, Status, PA_{0-7}, PB_{0-7}, PC_{0-5}) or the memory (RAM) portion.

The 8-bit address on the Address/Data lines, Chip Enable input CE or $\overline{CE}$, and IO/$\overline{M}$ are all latched on-chip at the falling edge of ALE.

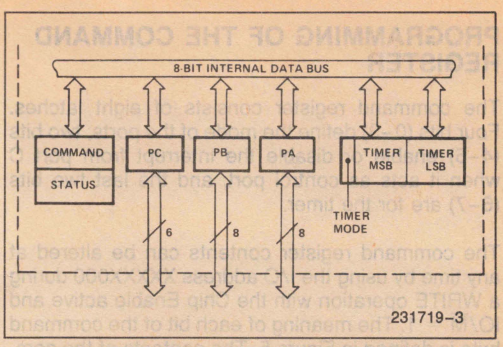
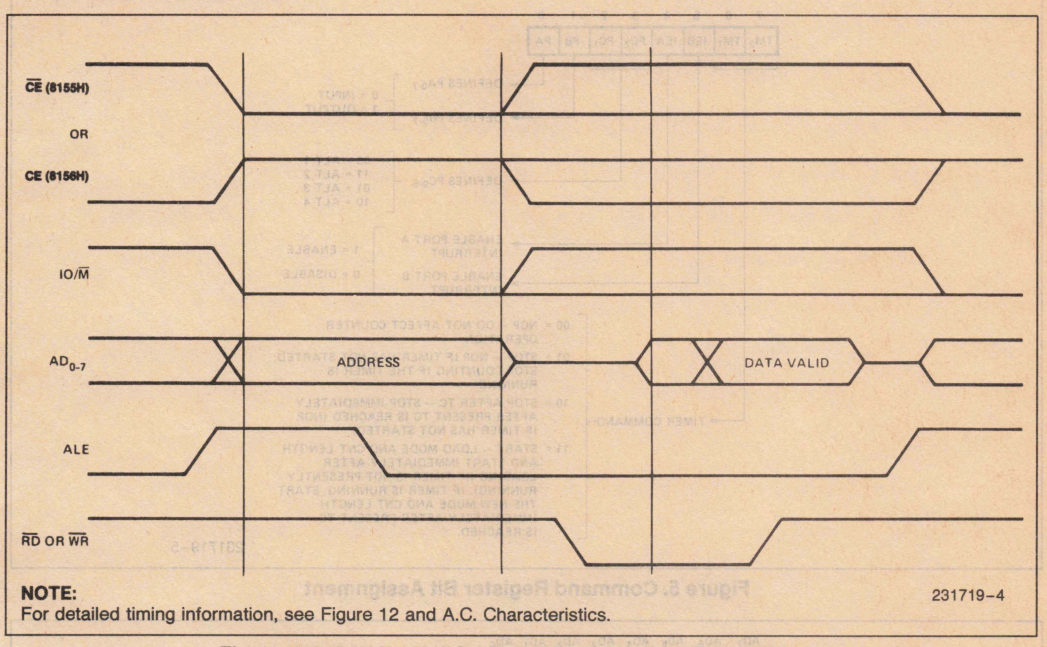


Figure 3. 8155H/8156H Internal Registers



NOTE:
For detailed timing information, see Figure 12 and A.C. Characteristics.

Figure 4. 8155H/8156H On-Board Memory Read/Write Cycle

PROGRAMMING OF THE COMMAND REGISTER

The command register consists of eight latches. Four bits (0–3) define the mode of the ports, two bits (4–5) enable or disable the interrupt from port C when it acts as control port, and the last two bits (6–7) are for the timer.

The command register contents can be altered at any time by using the I/O address XXXXX000 during a WRITE operation with the Chip Enable active and IO/$\overline{M}$ = 1. The meaning of each bit of the command byte is defined in Figure 5. The contents of the command register may never be read.

READING THE STATUS REGISTER

The status register consists of seven latches, one for each bit; six (0–5) for the status of the ports and one (6) for the status of the timer.

The status of the timer and the I/O section can be polled by reading the Status Register (Address XXXXX000). Status word format is shown in Figure 6. Note that you may never write to the status register since the command register shares the same I/O address and the command register is selected when a write to that address is issued.

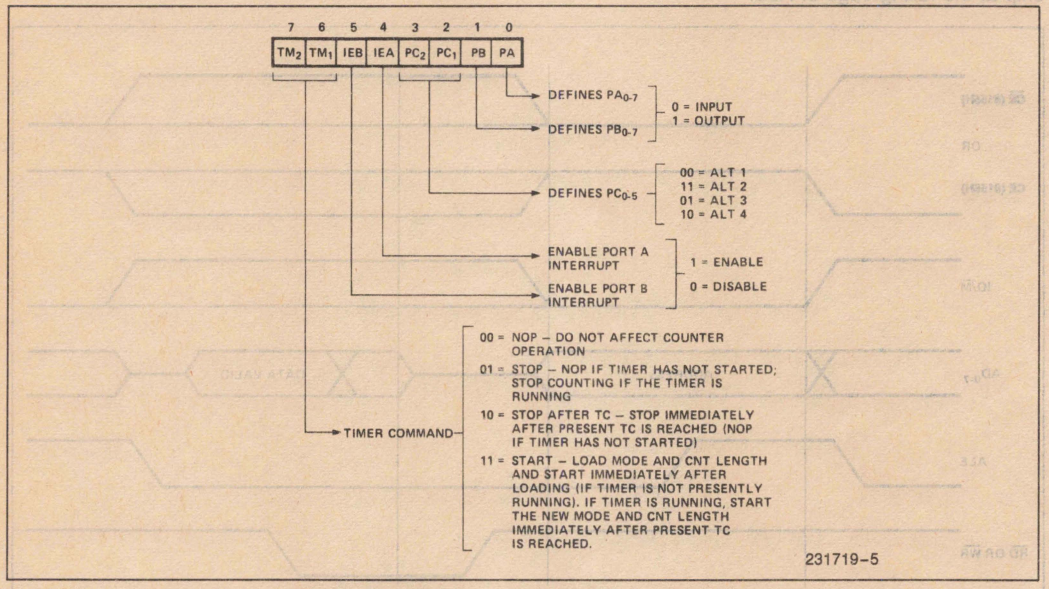


Figure 5. Command Register Bit Assignment

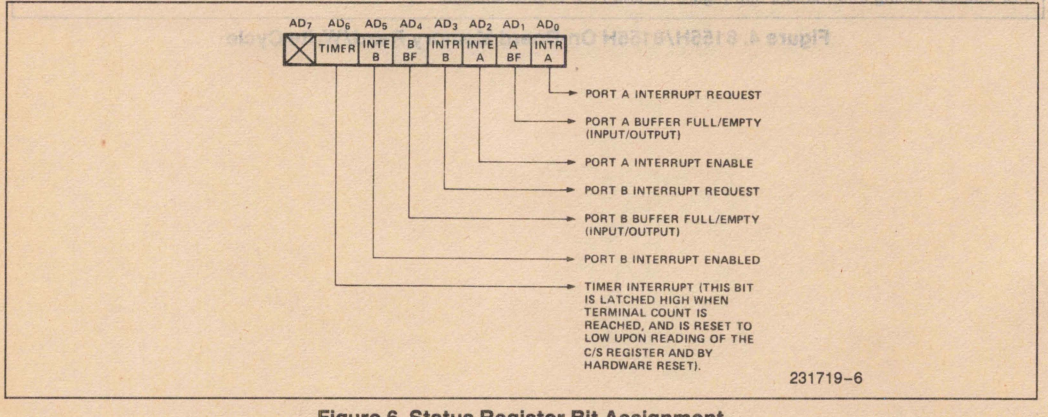


Figure 6. Status Register Bit Assignment

INPUT/OUTPUT SECTION

The I/O section of the 8155H/8156H consists of five registers: (see Figure 7.)

- **Command/Status Register (C/S)**—Both registers are assigned the address XXXXX000. The C/S address serves the dual purpose.
  When the C/S registers are selected during WRITE operation, a command is written into the command register. The contents of this register are *not* accessible through the pins.
  When the C/S (XXXXX000) is selected during a READ operation, the status information of the I/O ports and the timer becomes available on the AD_{0-7} lines.
- **PA Register**—This register can be programmed to be either input or output ports depending on the status of the contents of the C/S Register. Also depending on the command, this port can operate in either the basic mode or the strobed mode (see timing diagram). The I/O pins assigned in relation to this register are PA_{0-7}. The address of this register is XXXXX001.
- **PB Register**—This register functions the same as PA Register. The I/O pins assigned are PB_{0-7}. The address of this register is XXXXX010.
- **PC Register**—This register has the address XXXXX011 and contains only 6 bits. The 6 bits can be programmed to be either input ports, output ports or as control signals for PA and PB by properly programming the AD_2 and AD_3 bits of the C/S register.
  When PC_{0-5} is used as a control port, 3 bits are assigned for Port A and 3 for Port B. The first bit is an interrupt that the 8155H sends out. The second is an output signal indicating whether the buffer is full or empty, and the third is an input pin to accept a strobe for the strobed input mode. (See Table 2.)

When the 'C' port is programmed to either ALT3 or ALT4, the control signals for PA and PB are initialized as follows:

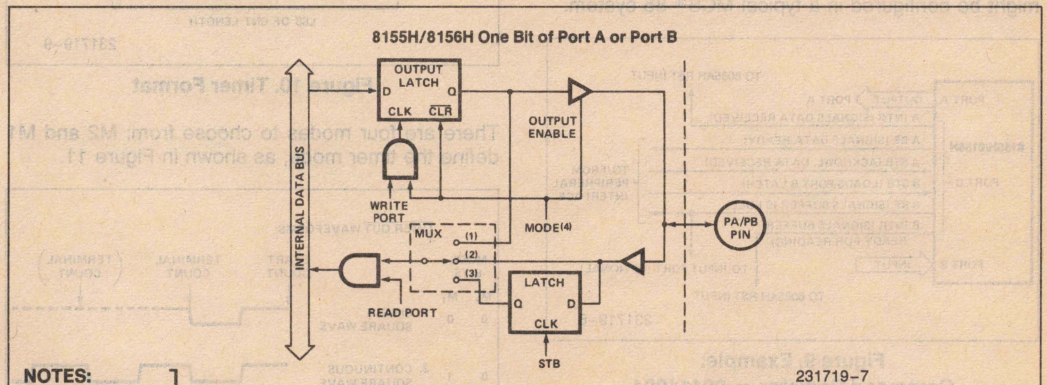
Control	Input Mode	Output Mode
BF	Low	Low
INTR	Low	High
STB	Input Control	Input Control

I/O Address†								Selection
A7	A6	A5	A4	A3	A2	A1	A0	
X	X	X	X	X	0	0	0	Interval Command/Status Register
X	X	X	X	X	0	0	1	General Purpose I/O Port A
X	X	X	X	X	0	1	0	General Purpose I/O Port B
X	X	X	X	X	0	1	1	Port C—General Purpose I/O or Control
X	X	X	X	X	1	0	0	Low-Order 8 bits of Timer Count
X	X	X	X	X	1	0	1	High 6 bits of Timer Count and 2 bits of Timer Mode

X: Don't Care.
†: I/O Address must be qualified by CE = 1 (8156H) or $\overline{CE}$ = 0 (8155H) and IO/$\overline{M}$ = 1 in order to select the appropriate register.

Figure 7. I/O Port and Timer Addressing Scheme

Figure 8 shows how I/O PORTS A and B are structured within the 8155H and 8156H:



NOTES:
(1) Output Mode
(2) Simple Input
(3) Strobed Input
Multiplexer Control

(4) = 1 for Output Mode
= 0 for Input Mode

READ Port = (IO/$\overline{M}$ = 1) • ($\overline{RD}$ = 0) • (CE Active) • (Port Address Selected)
WRITE Port = (IO/$\overline{M}$ = 1) • ($\overline{WR}$ = 0) • (CE Active) • (Port Address Selected)

Figure 8. 8155H/8156H Port Functions

Table 2. Port Control Assignment

Pin	ALT 1	ALT 2	ALT 3	ALT 4
PC0	Input Port	Output Port	A INTR (Port A Interrupt)	A INTR (Port A Interrupt)
PC1	Input Port	Output Port	A BF (Port A Buffer Full)	A BF (Port A Buffer Full)
PC2	Input Port	Output Port	A $\overline{STB}$ (Port A Strobe)	A $\overline{STB}$ (Port A Strobe)
PC3	Input Port	Output Port	Output Port	B INTR (Port B Interrupt)
PC4	Input Port	Output Port	Output Port	B BF (Port B Buffer Full)
PC5	Input Port	Output Port	Output Port	B $\overline{STB}$ (Port B Strobe)

Note in the diagram that when the I/O ports are programmed to be output ports, the contents of the output ports can still be read by a READ operation when appropriately addressed.

The outputs of the 8155H/8156H are "glitch-free" meaning that you can write a "1" to a bit position that was previously "1" and the level at the output pin will not change.

Note also that the output latch is cleared when the port enters the input mode. The output latch cannot be loaded by writing to the port if the port is in the input mode. The result is that each time a port mode is changed from input to output, the output pin will go low. When the 8155H/56H is RESET, the output latches are all cleared and all 3 ports enter the input mode.

When in the ALT 1 or ALT 2 modes, the bits of PORT C are structured like the diagram above in the simple input or output mode, respectively.

Reading from an input port with nothing connected to the pins will provide unpredictable results.

Figure 9 shows how the 8155H/8156H I/O ports might be configured in a typical MCS®-85 system.

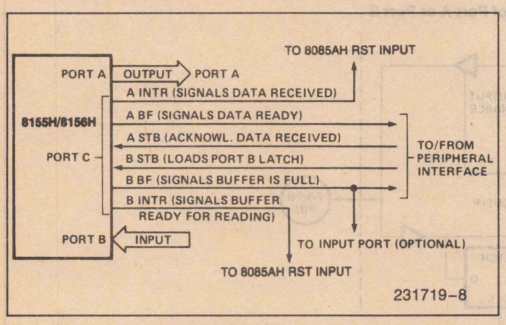


Figure 9. Example: Command Register = 00111001

TIMER SECTION

The time is a 14-bit down-counter that counts the TIMER IN pulses and provides either a square wave or pulse when terminal count (TC) is reached.

The timer has the I/O address XXXXX100 for the low order byte of the register and the I/O address XXXXX101 for the high order byte of the register. (See Figure 7.)

To program the timer, the COUNT LENGTH REG is loaded first, one byte at a time, by selecting the timer addresses. Bits 0–13 of the high order count register will specify the length of the next count and bits 14–15 of the high order register will specify the timer output mode (see Figure 10). The value loaded into the count length register can have any value from 2H through 3FFFH in Bits 0–13.

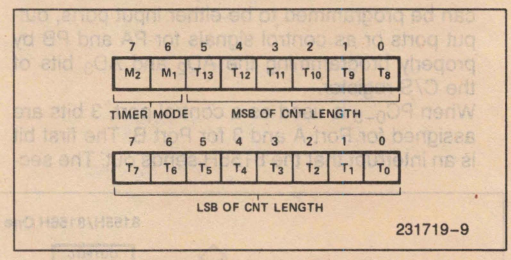


Figure 10. Timer Format

There are four modes to choose from: M2 and M1 define the timer mode, as shown in Figure 11.

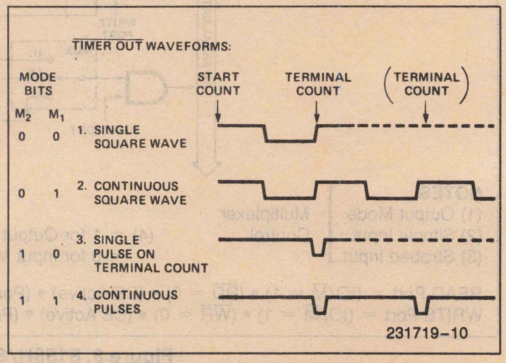


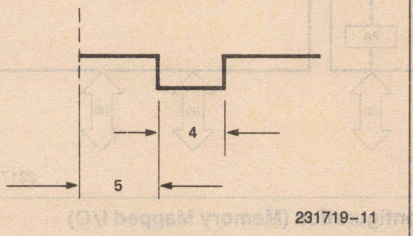
Figure 11. Timer Modes

Bits 6–7 (TM_2 and TM_1) of command register contents are used to start and stop the counter. There are four commands to choose from:

TM_2	TM_1	
0	0	NOP—Do not affect counter operation.
0	1	STOP—NOP if timer has not started; stop counting if the timer is running.
1	0	STOP AFTER TC—Stop immediately after present TC is reached (NOP if timer has not started)
1	1	START—Load mode and CNT length and start immediately after loading (if timer is not presently running). If timer is running, start the new mode and CNT length immediately after present TC is reached.

Note that while the counter is counting, you may load a new count and mode into the count length registers. Before the new count and mode will be used by the counter, you **must** issue a START command to the counter. This applies even though you may only want to change the count and use the previous mode.

In case of an odd-numbered count, the first half-cycle of the squarewave output, which is high, is one count longer than the second (low) half-cycle, as shown in Figure 12.



NOTE:
5 and 4 refer to the number of clocks in that time period.

Figure 12. Asymmetrical Square-Wave Output Resulting from Count of 9

The counter in the 8155H is not initialized to any particular mode or count when hardware RESET occurs, but RESET does *stop* the counting. Therefore, counting cannot begin following RESET until a START command is issued via the C/S register.

Please note that the timer circuit on the 8155H/8156H chip is designed to be a square-wave timer, not an event counter. To achieve this, it counts down by twos twice in completing one cycle. Thus, its registers do not contain values directly representing the number of TIMER IN pulses received. You cannot load an initial value of 1 into the count register and cause the timer to operate, as its terminal count value is 10 (binary) or 2 (decimal). (For the detection of single pulses, it is suggested that one of the hardware interrupt pins on the 8085AH be used.) After the timer has started counting down, the values residing in the count registers can be used to calculate the actual number of TIMER IN pulses required to complete the timer cycle if desired. To obtain the remaining count, perform the following operations in order:

1. Stop the count
2. Read in the 16-bit value from the count length registers
3. Reset the upper two mode bits
4. Reset the carry and rotate right one position all 16 bits through carry
5. If carry is set, add ½ of the full original count (½ full count—1 if full count is odd).

NOTE:
If you started with an odd count and you read the count length register before the third count pulse occurs, you will not be able to discern whether one or two counts has occurred. Regardless of this, the 8155H/56H always counts out the right number of pulses in generating the $\overline{\text{TIMER OUT}}$ waveforms.

8085AH MINIMUM SYSTEM CONFIGURATION

Figure 13a shows a minimum system using three chips, containing:

- 256 Bytes RAM
- 2K Bytes EPROM
- 38 I/O Pins
- 1 Interval Timer
- 4 Interrupt Levels

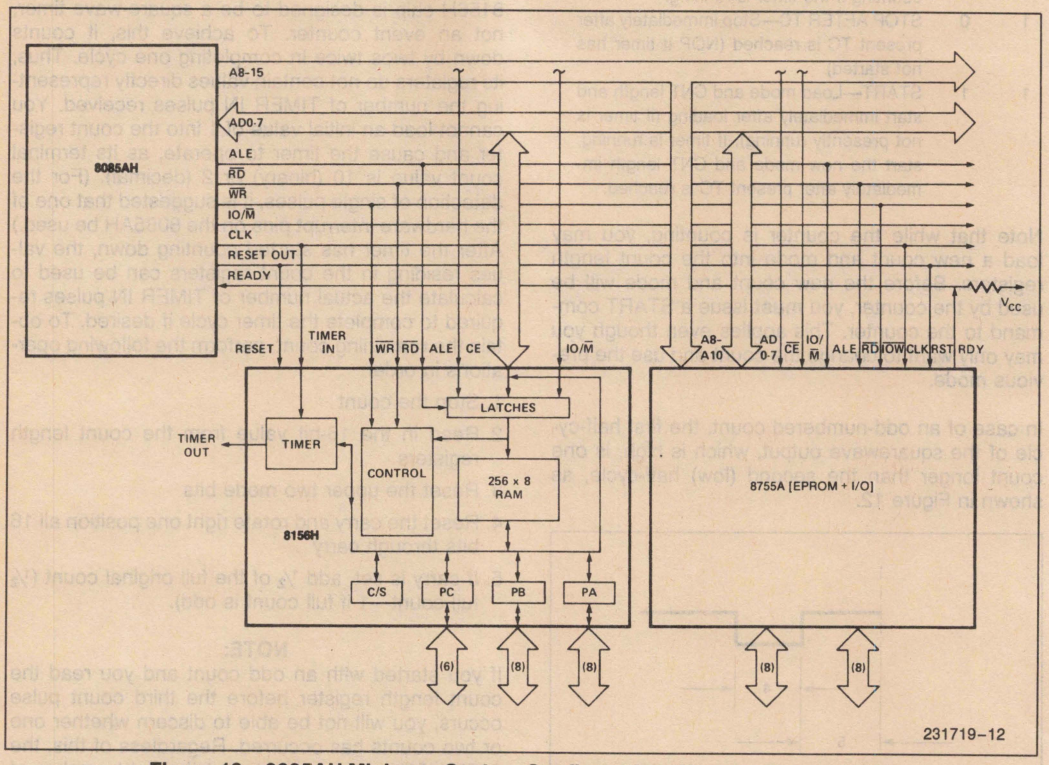


Figure 13a. 8085AH Minimum System Configuration (Memory Mapped I/O)

8088 FIVE CHIP SYSTEM

Figure 13b shows a five chip system containing:

- 1.25K Bytes RAM
- 2K Bytes EPROM
- 38 I/O Pins
- 1 Interval Timer
- 2 Interrupt Levels

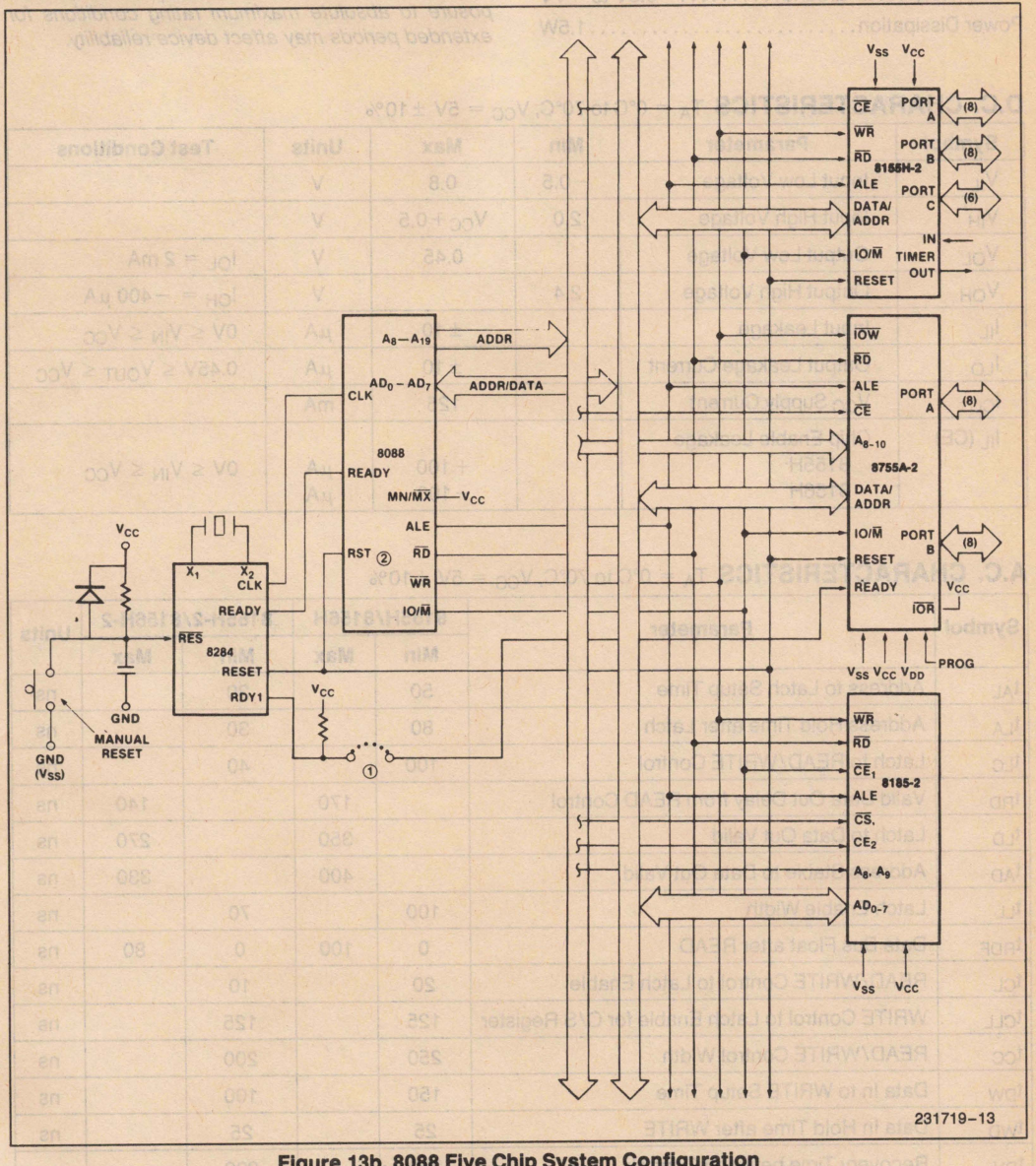


Figure 13b. 8088 Five Chip System Configuration

ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias ..... 0°C to +70°C
Storage Temperature ..... −65°C to +150°C
Voltage on Any Pin
with Respect to Ground ..... −0.5V to +7V
Power Dissipation 1.5W

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS T_A = 0°C to 70°C, V_{CC} = 5V ±10%

Symbol	Parameter	Min	Max	Units	Test Conditions
V_{IL}	Input Low Voltage	−0.5	0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC}+0.5	V	
V_{OL}	Output Low Voltage		0.45	V	I_{OL} = 2 mA
V_{OH}	Output High Voltage	2.4		V	I_{OH} = −400 μA
I_{IL}	Input Leakage		±10	μA	0V ≤ V_{IN} ≤ V_{CC}
I_{LO}	Output Leakage Current		±10	μA	0.45V ≤ V_{OUT} ≤ V_{CC}
I_{CC}	V_{CC} Supply Current		125	mA	
I_{IL} (CE)	Chip Enable Leakage 8155H 8156H		 +100 −100	 μA μA	0V ≤ V_{IN} ≤ V_{CC}

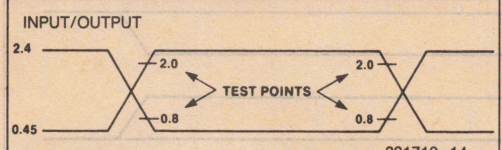
A.C. CHARACTERISTICS T_A = 0°C to 70°C, V_{CC} = 5V ±10%

Symbol	Parameter	8155H/8156H		8155H-2/8156H-2		Units
		Min	Max	Min	Max	
t_{AL}	Address to Latch Setup Time	50		30		ns
t_{LA}	Address Hold Time after Latch	80		30		ns
t_{LC}	Latch to READ/WRITE Control	100		40		ns
t_{RD}	Valid Data Out Delay from READ Control		170		140	ns
t_{LD}	Latch to Data Out Valid		350		270	ns
t_{AD}	Address Stable to Data Out Valid		400		330	ns
t_{LL}	Latch Enable Width	100		70		ns
t_{RDF}	Data Bus Float after READ	0	100	0	80	ns
t_{CL}	READ/WRITE Control to Latch Enable	20		10		ns
t_{CLL}	WRITE Control to Latch Enable for C/S Register	125		125		ns
t_{CC}	READ/WRITE Control Width	250		200		ns
t_{DW}	Data In to WRITE Setup Time	150		100		ns
t_{WD}	Data In Hold Time after WRITE	25		25		ns
t_{RV}	Recovery Time between Controls	300		200		ns
t_{WP}	WRITE to Port Output		400		300	ns

A.C. CHARACTERISTICS T_A = 0°C to 70°C, V_{CC} = 5V ±10% (Continued)

Symbol	Parameter	8155H/8156H		8155H-2/8156H-2		Units
		Min	Max	Min	Max	
t_{PR}	Port Input Setup Time	70		50		ns
t_{RP}	Port Input Hold Time	50		10		ns
t_{SBF}	Strobe to Buffer Full		400		300	ns
t_{SS}	Strobe Width	200		150		ns
t_{RBE}	READ to Buffer Empty		400		300	ns
t_{SI}	Strobe to INTR On		400		300	ns
t_{RDI}	READ to INTR Off		400		300	ns
t_{PSS}	Port Setup Time to Strobe	50		0		ns
t_{PHS}	Port Hold Time After Strobe	120		100		ns
t_{SBE}	Strobe to Buffer Empty		400		300	ns
t_{WBF}	WRITE to Buffer Full		400		300	ns
t_{WI}	WRITE to INTR Off		400		300	ns
t_{TL}	TIMER-IN to $\overline{\text{TIMER-OUT}}$ Low		400		300	ns
t_{TH}	TIMER-IN to $\overline{\text{TIMER-OUT}}$ High		400		300	ns
t_{RDE}	Data Bus Enable from READ Control	10		10		ns
t_1	TIMER-IN Low Time	80		40		ns
t_2	TIMER-IN High Time	120		70		ns
t_{WT}	WRITE to TIMER-IN (for writes which start counting)	360		200		ns

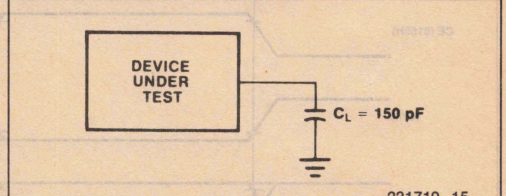
A.C. TESTING INPUT, OUTPUT WAVEFORM



231719-14

A.C. testing: inputs are driven at 2.4V for a logic "1" and 0.45V for a logic "0". Timing measurements are made at 2.0V for a logic "1" and 0.8V for a logic "0".

A.C. TESTING LOAD CIRCUIT

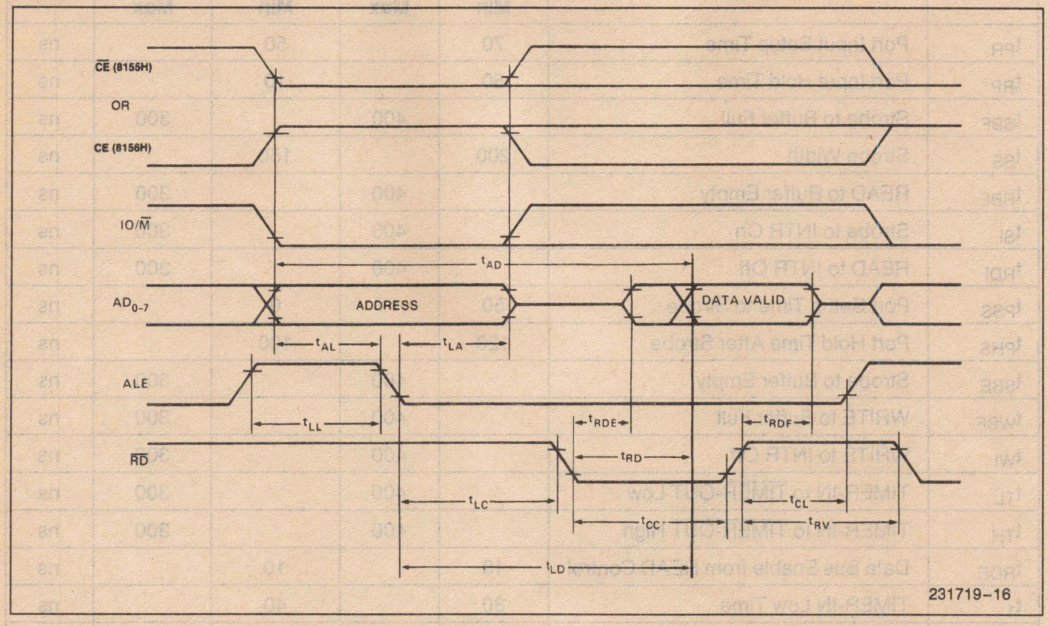


231719-15

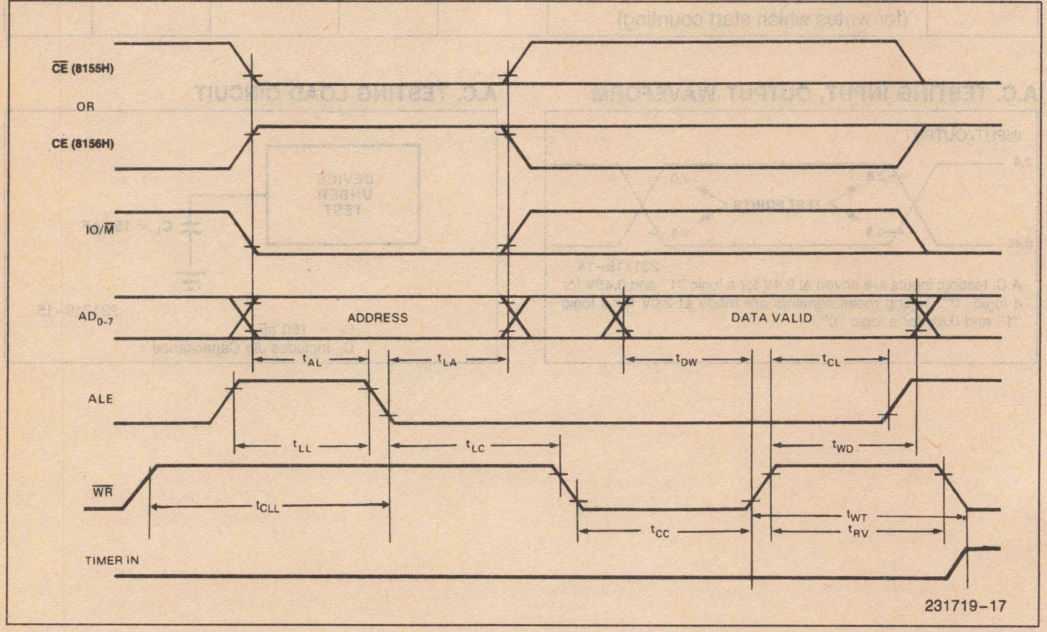
C_L = 150 pF
C_L Includes Jig Capacitance

WAVEFORMS

READ

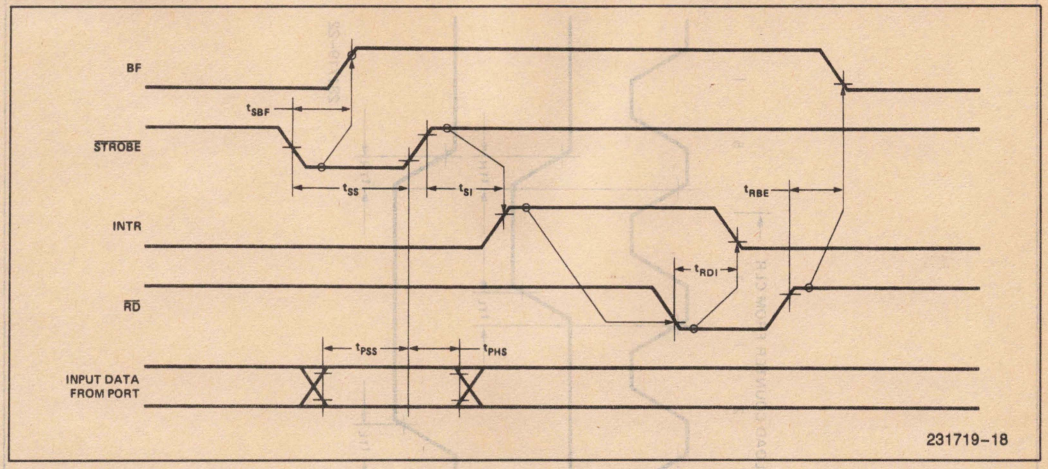


WRITE

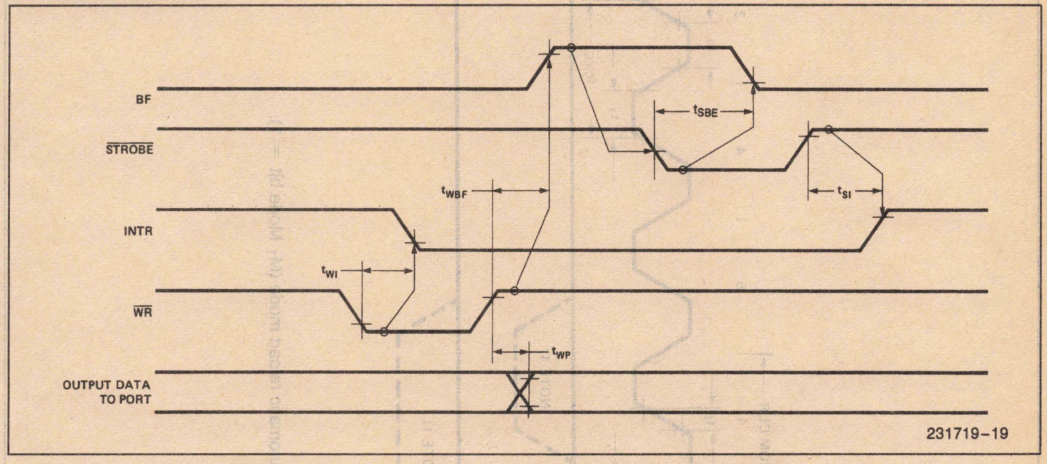


WAVEFORMS (Continued)

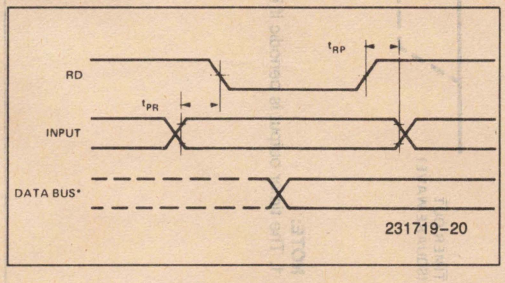
STROBED INPUT



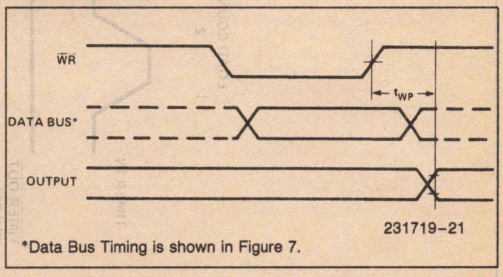
STROBED OUTPUT



BASIC INPUT



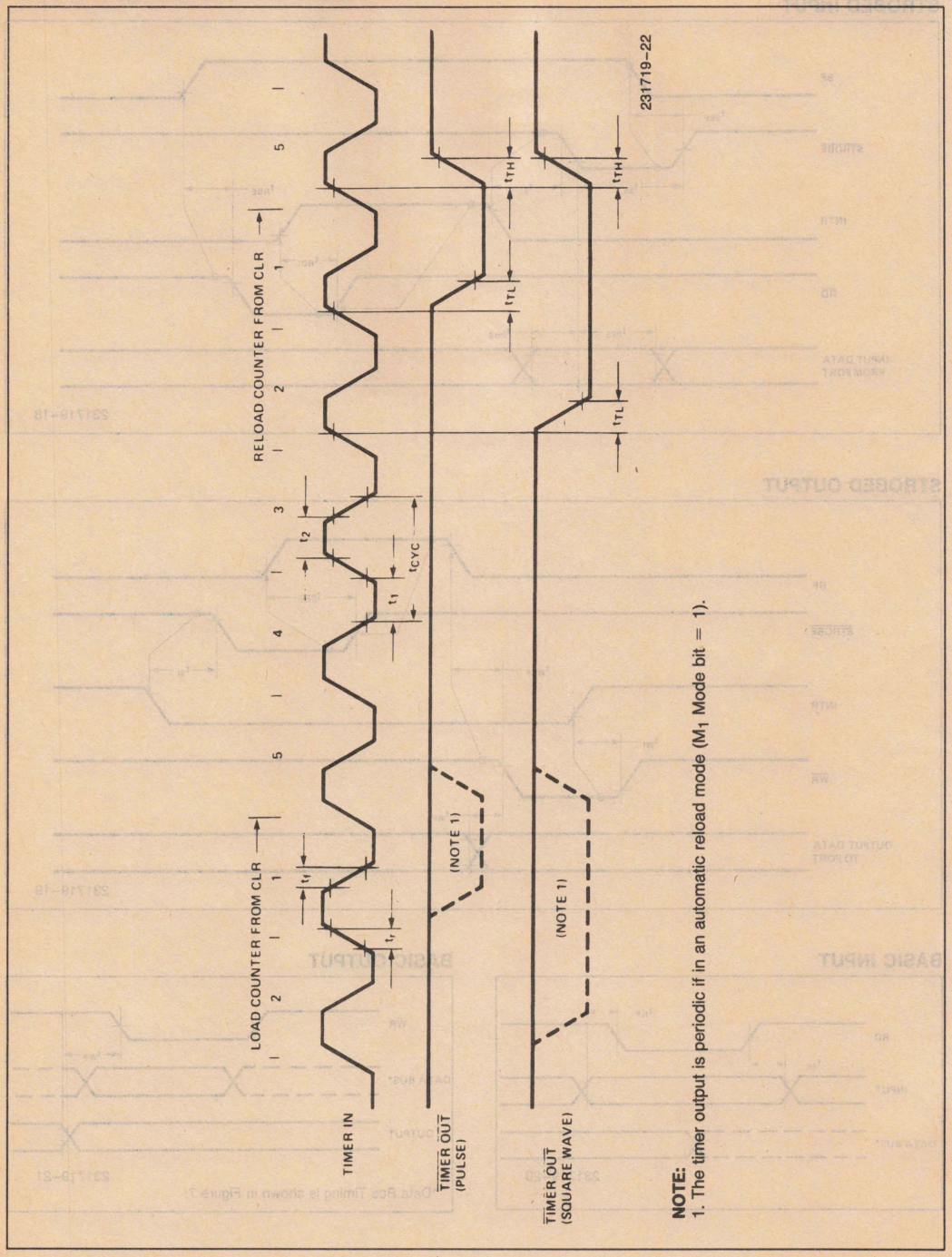
BASIC OUTPUT



*Data Bus Timing is shown in Figure 7.

WAVEFORMS (Continued)

TIMER OUTPUT COUNTDOWN FROM 5 TO 1



NOTE:
1. The timer output is periodic if in an automatic reload mode (M_1 Mode bit = 1).

8185/8185-2
1024 x 8-BIT STATIC RAM FOR MCS®-85

- **Multiplexed Address and Data Bus**
- **Directly Compatible with 8085AH and 8088 Microprocessors**
- **Low Operating Power Dissipation**
- **Low Standby Power Dissipation**
- **Single +5V Supply**
- **High Density 18-Pin Package**

The Intel 8185 is an 8192-bit static random access memory (RAM) organized as 1024 words by 8-bits using N-channel Silicon-Gate MOS technology. The multiplexed address and data bus allows the 8185 to interface directly to the 8085AH and 8088 microprocessors to provide a maximum level of system integration.

The low standby power dissipation minimizes system power requirements when the 8185 is disabled.

The 8185-2 is a high-speed selected version of the 8185 that is compatible with the 5 MHz 8085AH-2 and the 5 MHz 8088.

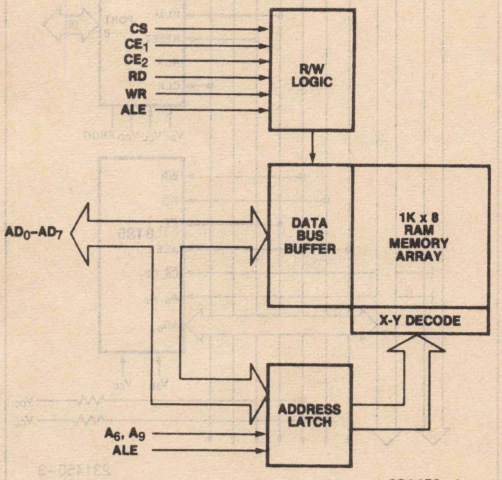


Figure 1. Block Diagram

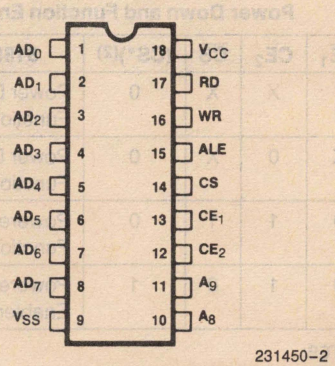


Figure 2. Pin Configuration

Pin Names

AD_0–AD_7	Address/Data Lines
A_8, A_9	Address Lines
CS	Chip Select
CE_1	Chip Enable (IO/M)
CE_2	Chip Enable
ALE	Address Latch Enable
WR	Write Enable

November 1986
Order Number: 231450-002

FUNCTIONAL DESCRIPTION

The 8185 has been designed to provide for direct interface to the multiplexed bus structure and bus timing of the 8085A microprocessor.

At the beginning of an 8185 memory access cycle, the 8-bit address on AD_{0-7}, A_8 and A_9, and the status of $\overline{CE_1}$ and CE_2 are all latched internally in the 8185 by the falling edge of ALE. If the latched status of both $\overline{CE_1}$ and CE_2 are active, the 8185 powers itself up, but no action occurs until the $\overline{CS}$ line goes low and the appropriate $\overline{RD}$ or $\overline{WR}$ control signal input is activated.

The $\overline{CS}$ input is not latched by the 8185 in order to allow the maximum amount of time for address decoding in selecting the 8185 chip. Maximum power consumption savings will occur, however, only when $\overline{CE_1}$ and CE_2 are activated selectively to power down the 8185 when it is not in use. A possible connection would be to wire the 8085A's IO/$\overline{M}$ line to the 8185's $\overline{CE_1}$ input, thereby keeping the 8185 powered down during I/O and interrupt cycles.

Table 1. Truth Table for Power Down and Function Enable

$\overline{CE_1}$	CE_2	$\overline{CS}$	(CS*)(2)	8185 Status
1	X	X	0	Power Down and Function Disable(1)
X	0	X	0	Power Down and Function Disable(1)
0	1	1	0	Powered Up and Function Disable(1)
0	1	0	1	Powered Up and Enabled

NOTES:
X = Don't Care.
1: Function Disable implies Data Bus in high impedance state and not writing.
2: CS* = $(\overline{CE_1} = 0) \times (CE_2 = 1) \times (\overline{CS} = 0)$.
CS* = 1 signifies all chip enables and chip select active.

Table 2. Truth Table for Control and Data Bus Pin Status

(CS*)	$\overline{RD}$	$\overline{WR}$	AD_{0-7} During Data Portion of Cycle	8185 Function
0	X	X	Hi-Impedance	No Function
1	0	1	Data from Memory	Read
1	1	0	Data to Memory	Write
1	1	1	Hi-Impedance	Reading, but not Driving Data Bus

NOTE:
X = Don't Care.

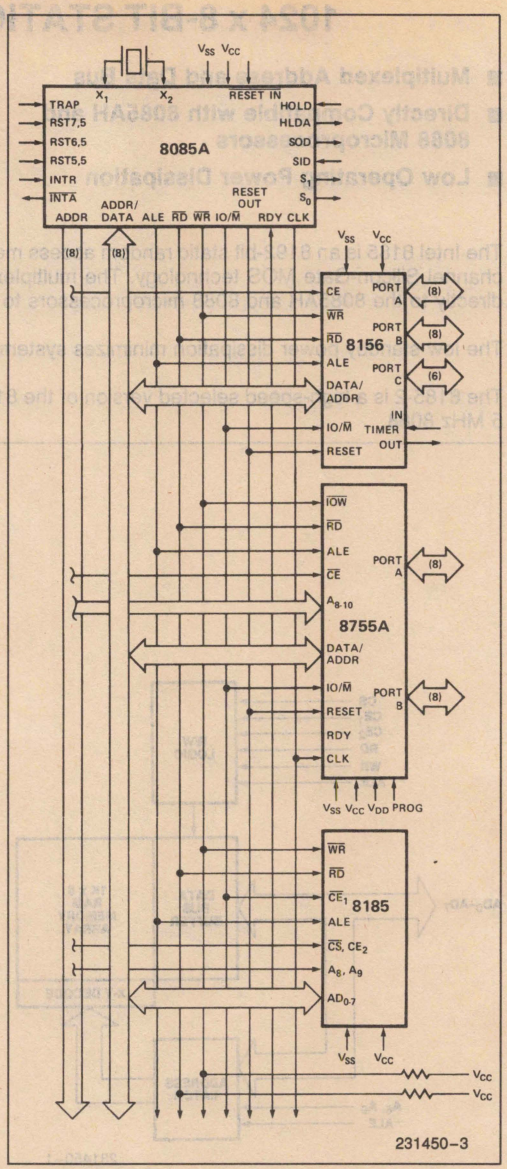


Figure 3. 8185 in an MCS®-85 System

4 Chips:
2K Bytes EPROM
1.25K Bytes RAM
38 I/O Lines
1 Counter/Timer
2 Serial I/O Lines
5 Interrupt Inputs

iAPX 88 FIVE CHIP SYSTEM:

- 1.25K Bytes RAM
- 2K Bytes EPROM
- 38 I/O Pins
- 1 Internal Timer
- 2 Interrupt Levels

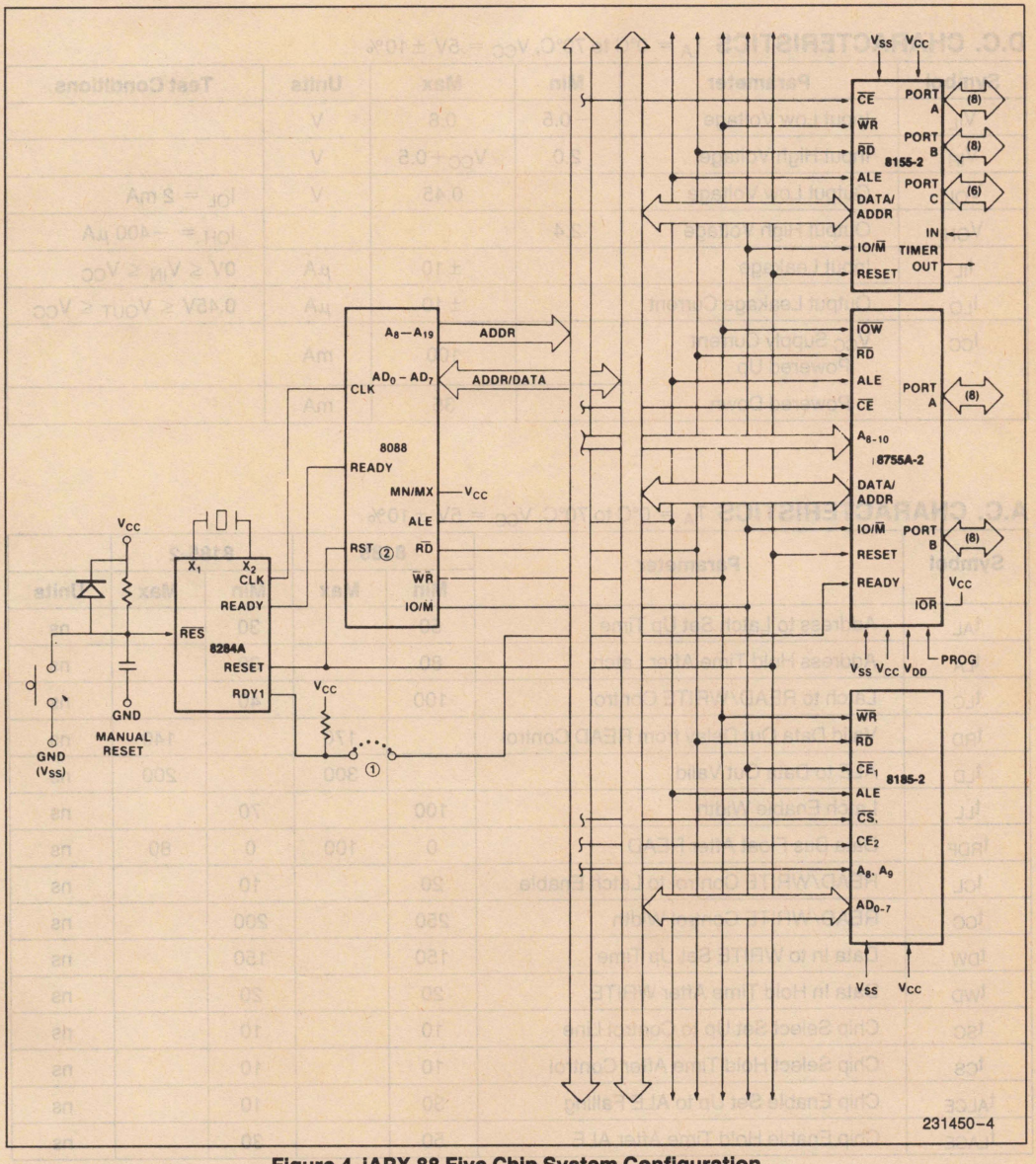


Figure 4. iAPX 88 Five Chip System Configuration

ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias .....0°C to +70°C
Storage Temperature .....−65°C to +150°C
Voltage on Any Pin
with Respect to Ground.....−0.5V to +7V
Power Dissipation.....1.5W

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

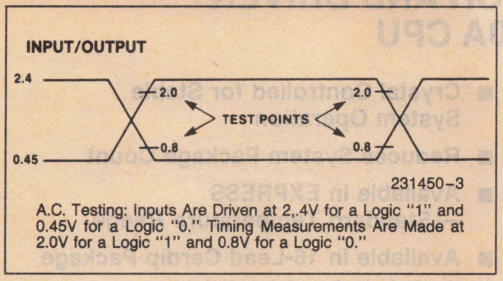
D.C. CHARACTERISTICS T_A = 0°C to 70°C, V_{CC} = 5V ±10%

Symbol	Parameter	Min	Max	Units	Test Conditions
V_{IL}	Input Low Voltage	−0.5	0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC}+0.5	V	
V_{OL}	Output Low Voltage		0.45	V	I_{OL} = 2 mA
V_{OH}	Output High Voltage	2.4			I_{OH} = −400 μA
I_{IL}	Input Leakage		±10	μA	0V ≤ V_{IN} ≤ V_{CC}
I_{LO}	Output Leakage Current		±10	μA	0.45V ≤ V_{OUT} ≤ V_{CC}
I_{CC}	V_{CC} Supply Current Powered Up		100	mA	
	Powered Down		35	mA	

A.C. CHARACTERISTICS T_A = 0°C to 70°C, V_{CC} = 5V ±10%

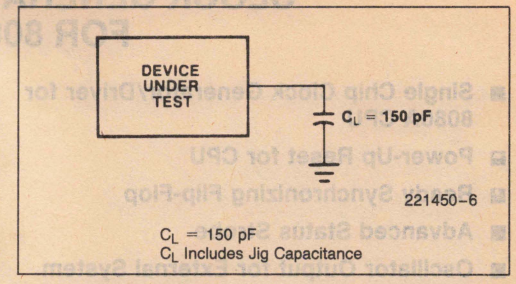
Symbol	Parameter	8185		8185-2		
		Min	Max	Min	Max	Units
t_{AL}	Address to Latch Set Up Time	50		30		ns
t_{LA}	Address Hold Time After Latch	80		30		ns
t_{LC}	Latch to READ/WRITE Control	100		40		ns
t_{RD}	Valid Data Out Delay from READ Control		170		140	ns
t_{LD}	ALE to Data Out Valid		300		200	ns
t_{LL}	Latch Enable Width	100		70		ns
t_{RDF}	Data Bus Float After READ	0	100	0	80	ns
t_{CL}	READ/WRITE Control to Latch Enable	20		10		ns
t_{CC}	READ/WRITE Control Width	250		200		ns
t_{DW}	Data In to WRITE Set Up Time	150		150		ns
t_{WD}	Data In Hold Time After WRITE	20		20		ns
t_{SC}	Chip Select Set Up to Control Line	10		10		ns
t_{CS}	Chip Select Hold Time After Control	10		10		ns
t_{ALCE}	Chip Enable Set Up to ALE Falling	30		10		ns
t_{LACE}	Chip Enable Hold Time After ALE	50		30		ns

A.C. TESTING INPUT, OUTPUT WAVEFORM



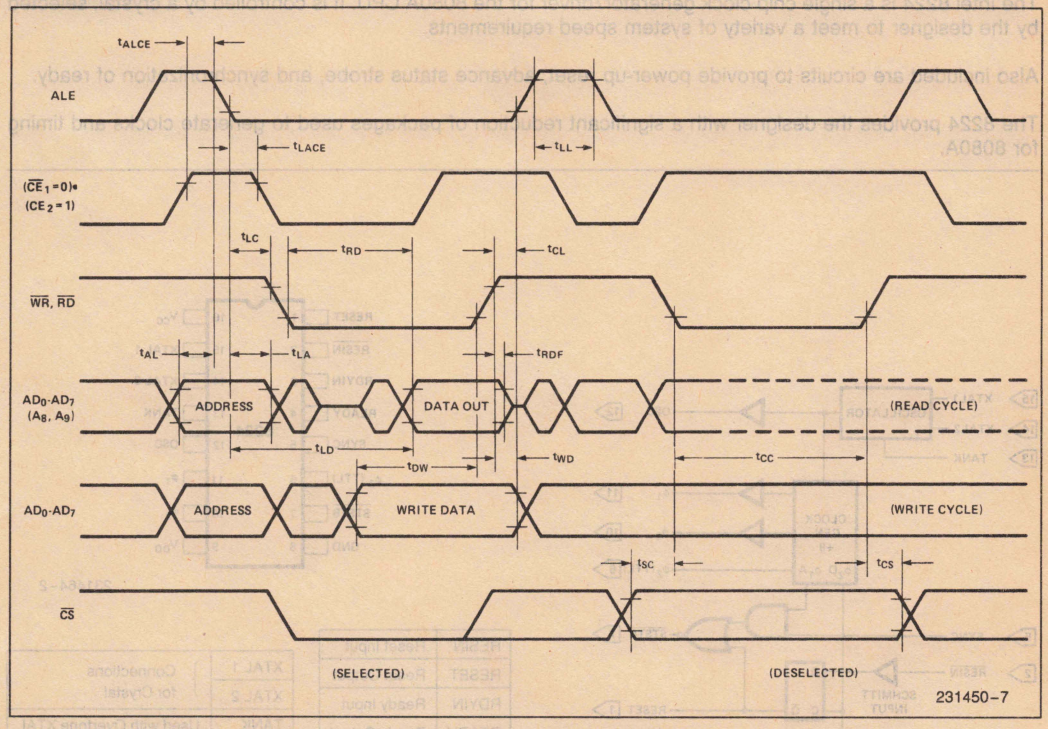
A.C. Testing: Inputs Are Driven at 2,.4V for a Logic "1" and 0.45V for a Logic "0." Timing Measurements Are Made at 2.0V for a Logic "1" and 0.8V for a Logic "0."

A.C. TESTING LOAD CIRCUIT



C_L = 150 pF
C_L Includes Jig Capacitance

WAVEFORM



8224
CLOCK GENERATOR AND DRIVER FOR 8080A CPU

- **Single Chip Clock Generator/Driver for 8080A CPU**
- **Power-Up Reset for CPU**
- **Ready Synchronizing Flip-Flop**
- **Advanced Status Strobe**
- **Oscillator Output for External System Timing**
- **Crystal Controlled for Stable System Operation**
- **Reduces System Package Count**
- **Available in EXPRESS — Standard Temperature Range**
- **Available in 16-Lead Cerdip Package**
  (See Packaging Spec, Order #231369)

The Intel 8224 is a single chip clock generator/driver for the 8080A CPU. It is controlled by a crystal, selected by the designer to meet a variety of system speed requirements.

Also included are circuits to provide power-up reset, advance status strobe, and synchronization of ready.

The 8224 provides the designer with a significant reduction of packages used to generate clocks and timing for 8080A.

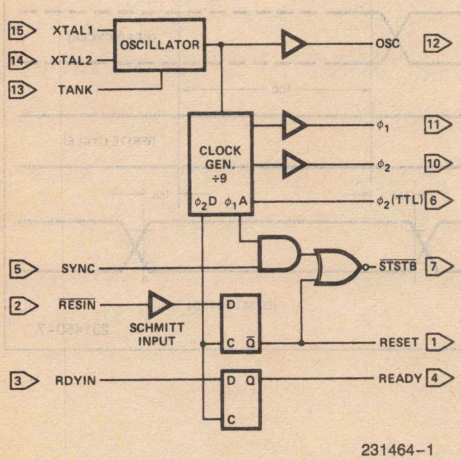
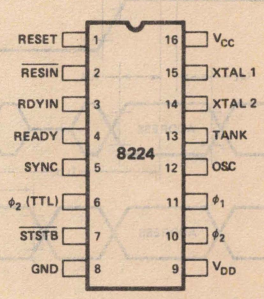


Figure 1. Block Diagram



$\overline{\text{RESIN}}$	Reset Input
RESET	Reset Output
RDYIN	Ready Input
READY	Ready Output
SYNC	Sync Input
$\overline{\text{STSTB}}$	Status STB (Active Low)
ϕ_1	8080 Clocks
ϕ_2	

XTAL 1	Connections for Crystal
XTAL 2	
TANK	Used with Overtone XTAL
OSC	Oscillator Output
ϕ_2 (TTL)	ϕ_2 CLK (TTL Level)
V_{CC}	+5V
V_{DD}	+12V
GND	0V

Figure 2. Pin Configuration

November 1986
Order Number: 231464-001

ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias .....0°C to +70°C
Storage Temperature .....−65°C to +150°C
Supply Voltage, V_{CC} .....−0.5V to +7V
Supply Voltage, V_{DD} .....−0.5V to +13.5V
Input Voltage .....−1.5V to +7V
Output Current .....100 mA

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS

T_A = 0°C to +70°C, V_{CC} = +5.0V ±5%, V_{DD} = +12V ±5%

Symbol	Parameter	Limits			Units	Test Conditions
		Min	Typ	Max		
I_F	Input Current Loading			−0.25	mA	V_F = 0.45V
I_R	Input Leakage Current			10	μA	V_R = 5.25V
V_C	Input Forward Clamp Voltage			1.0	V	I_C = −5 mA
V_{IL}	Input "Low" Voltage			0.8	V	V_{CC} = 5.0V
V_{IH}	Input "High" Voltage	2.6			V	Reset Input
		2.0			V	All Other Inputs
$V_{IH}-V_{IL}$	RESIN Input Hysteresis	0.25			V	V_{CC} = 5.0V
V_{OL}	Output "Low" Voltage			0.45	V	(ϕ_1, ϕ_2), Ready, Reset, $\overline{STSTB}$ I_{OL} = 2.5 mA
				0.45	V	All Other Outputs I_{OL} = 15 mA
V_{OH}	Output "High" Voltage ϕ_1, ϕ_2	9.4			V	I_{OH} = −100 μA
	READY, RESET	3.6			V	I_{OH} = −100 μA
	All Other Outputs	2.4			V	I_{OH} = −1 mA
I_{CC}	Power Supply Current			115	mA	
I_{DD}	Power Supply Current			12	mA	

NOTE:
1. For crystal frequencies of 18 MHz connect 510Ω resistors between the X1 input and ground as well as the X2 input and ground to prevent oscillation at harmonic frequencies.

Crystal Requirements

Tolerance: 0.005% at 0°C–70°C
Resonance: Series (Fundamental)*
Load Capacitance: 20 pF–35 pF
Equivalent Resistance: 75Ω–20Ω

Power Dissipation (Min): 4 mW

***NOTE:**
With tank circuit use 3rd overtone mode.

A.C. CHARACTERISTICS

Symbol	Parameter	Limits			Units	Test Conditions
		Min	Typ	Max		
$t_{\phi 1}$	ϕ_1 Pulse Width	$\frac{2tcy}{9} - 20$ ns			ns	C_L = 20 pF to 50 pF
$t_{\phi 2}$	ϕ_2 Pulse Width	$\frac{5tcy}{9} - 35$ ns				
t_{D1}	ϕ_1 to ϕ_2 Delay	0				
t_{D2}	ϕ_2 to ϕ_1 Delay	$\frac{2tcy}{9} - 14$ ns				
t_{D3}	ϕ_1 to ϕ_2 Delay	$\frac{2tcy}{9}$		$\frac{2tcy}{9} + 20$ ns		
t_R	ϕ_1 and ϕ_2 Rise Time			20		
t_F	ϕ_1 and ϕ_2 Fall Time			20		
$t_{D\phi 2}$	ϕ_2 to ϕ_2 (TTL) Delay	−5		+15	ns	ϕ_2 TTL, CL = 30 R_1 = 300Ω R_2 = 600Ω
t_{DSS}	ϕ_2 to $\overline{STSTB}$ Delay	$\frac{6tcy}{9} - 30$ ns		$\frac{6tcy}{9}$	ns	$\overline{STSTB}$, C_L = 15 pF R_1 = 2K R_2 = 4K
t_{PW}	$\overline{STSTB}$ Pulse Width	$\frac{tcy}{9} - 15$ ns			ns	
t_{DRS}	RDYIN Setup Time to Status Strobe	$50\text{ ns} - \frac{4tcy}{9}$				
t_{DRH}	RDYIN Hold Time after $\overline{STSTB}$	$\frac{4tcy}{9}$				
t_{DR}	RDYIN or RESIN to ϕ_2 Delay	$\frac{4tcy}{9} - 25$ ns			ns	Ready & Reset C_L = 10 pF R_1 = 2K R_2 = 4K
t_{CLK}	CLK Period		$\frac{tcy}{9}$		ns	
f_{max}	Maximum Oscillating Frequency			27	MHz	
C_{in}	Input Capacitance			8	pF	V_{CC} = +5.0V V_{DD} = +12V V_{BIAS} = 2.5V f = 1 MHz

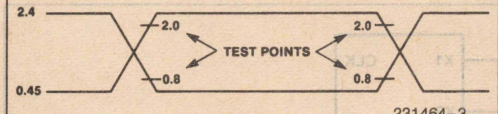
NOTE:
These formulas are based on the internal workings of the part and intended for customer convenience. Actual testing of the part is done at t_{cy} = 488.28 ns.

A.C. CHARACTERISTICS (Continued)

For t_{CY} = 488.28 ns; T_A = 0°C to 70°C, V_{CC} = +5V ±5%, V_{DD} = +12V ±5%

Symbol	Parameter	Limits			Units	Test Conditions
		Min	Typ	Max		
$t_{\phi 1}$	ϕ_1 Pulse Width	89			ns	t_{CY} = 488.28 ns
$t_{\phi 2}$	ϕ_2 Pulse Width	236			ns	
t_{D1}	Delay ϕ_1 to ϕ_2	0			ns	
t_{D2}	Delay ϕ_2 to ϕ_1	95			ns	ϕ_1 & ϕ_2 Loaded to C_L = 20 pF to 50 pF
t_{D3}	Delay ϕ_1 to ϕ_2 Leading Edges	109		129	ns	
t_r	Output Rise Time			20	ns	
t_f	Output Fall Time			20	ns	
t_{DSS}	ϕ_2 to $\overline{STSTB}$ Delay	296		326	ns	
$t_{D\phi 2}$	ϕ_2 to ϕ_2 (TTL) Delay	−5		+15	ns	
t_{PW}	Status Strobe Pulse Width	40			ns	Ready & Reset Loaded to 2 mA/10 pF All measurements referenced to 1.5V unless specified otherwise.
t_{DRS}	RDYIN Setup Time to $\overline{STSTB}$	−167			ns	
t_{DRH}	RDYIN Hold Time after $\overline{STSTB}$	217			ns	
t_{DR}	READY or RESET to ϕ_2 Delay	192			ns	
f_{MAX}	Oscillator Frequency			18.432	MHz	

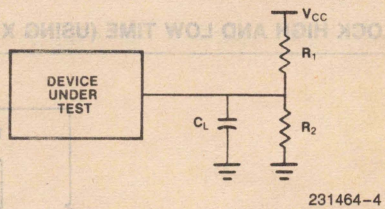
A.C. TESTING, INPUT, OUTPUT WAVEFORM



231464-3

A.C. Testing: Inputs are driven at 2.4V for a logic "1" and 0.45V for a logic "0". Timing measurements are made at 2.0V for a logic "1" and 0.8V for a logic "0" (unless otherwise noted).

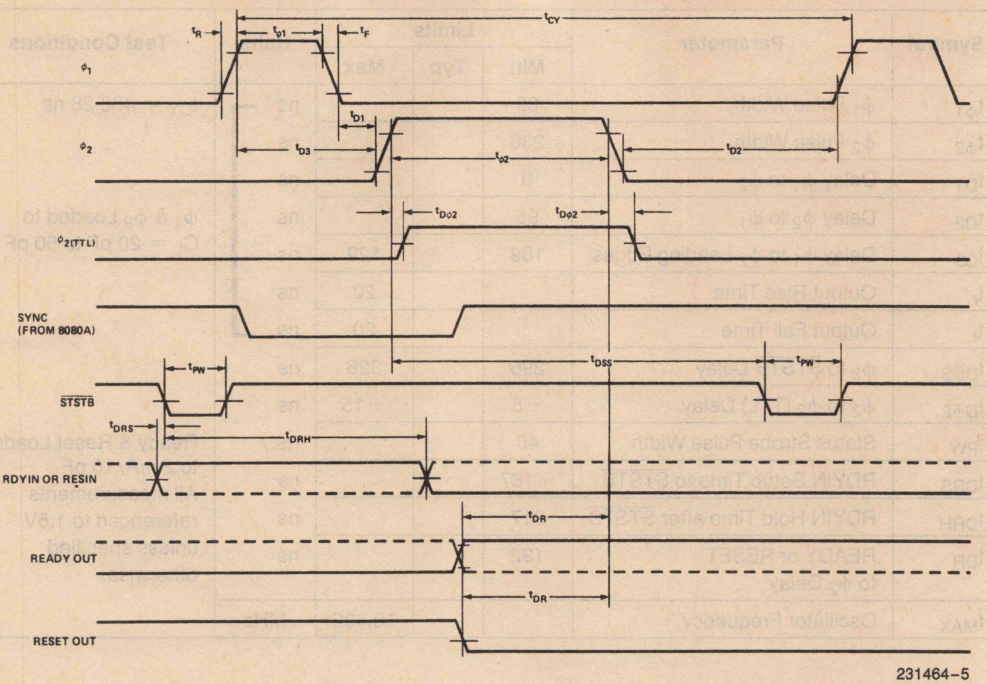
A.C. TESTING LOAD CIRCUIT



231464-4

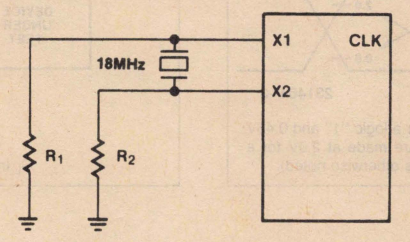
C_L Includes Jig Capacitance

WAVEFORMS



VOLTAGE MEASUREMENT POINTS: ϕ_1, ϕ_2 Logic "0" = 1.0V, Logic "1" = 8.0V. All other signals measured at 1.5V.

CLOCK HIGH AND LOW TIME (USING X1, X2)



8228/8238
SYSTEM CONTROLLER AND BUS DRIVER FOR 8080A CPU

- **Single Chip System Control for MCS®-80 Systems**
- **Built-In Bidirectional Bus Driver for Data Bus Isolation**
- **Allows the Use of Multiple Byte Instructions (e.g. CALL) for Interrupt Acknowledge**
- **Reduces System Package Count**
- **User Selected Single Level Interrupt Vector (RST 7)**
- **8238 Has Advanced IOW/MEMW for Large System Timing Control**
- **Available in EXPRESS — Standard Temperature Range**
- **Available in 28-Lead Cerdip and Plastic Packages**

(See Packaging Spec, Order #231369)

The Intel® 8228 is a single chip system controller and bus driver for MCS®-80. It generates all signals required to directly interface MCS-80 family RAM, ROM, and I/O components.

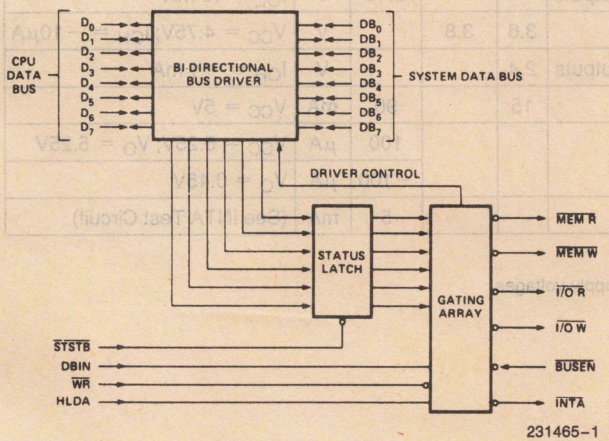
A bidirectional bus driver is included to provide high system TTL fan-out. It also provides isolation of the 8080 data bus from memory and I/O. This allows for the optimization of control signals, enabling the systems designer to use slower memory and I/O. The isolation of the bus driver also provides for enhanced system noise immunity.

A user selected single level interrupt vector (RST 7) is provided to simplify real time, interrupt driven, small system requirements. The 8228 also generates the correct control signals to allow the use of multiple byte instructions (e.g., CALL) in response to an interrupt acknowledge by the 8080A. This feature permits large, interrupt driven systems to have an unlimited number of interrupt levels.

The 8228 is designed to support a wide variety of system bus structures and also reduce system package count for cost effective, reliable design of MCS-80 systems.

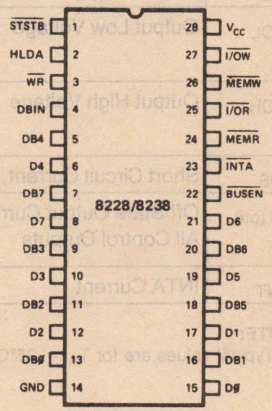
NOTE:

The specifications for the 3228/3238 are identical with those for the 8228/8238.



231465-1

Figure 1. Block Diagram



231465-2

D7-D0	Data Bus (8080 Side)
DB7-DB0	Data Bus (System Side)
I/$\overline{OR}$	I/O Read
I/$\overline{OW}$	I/O Write
$\overline{MEMR}$	Memory Read
$\overline{MEMW}$	Memory Write
DBIN	DBIN (from 8080)

INTA	Interrupt Acknowledge
HLDA	HLDA (from 8080)
$\overline{WR}$	WR (from 8080)
$\overline{BUSEN}$	Bus Enable Input
$\overline{STSTB}$	Status Strobe (from 8224)
V_{CC}	+5V
GND	0 Volts

Figure 2. Pin Configuration

November 1986
Order Number: 231465-001

ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias ..... −0°C to 70°C
Storage Temperature ..... −65°C to +150°C
Supply Voltage, V_{CC} ..... −0.5V to +7V
Input Voltage ..... −1.5 to +7V
Output Current ..... 100 mA

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS T_A = 0°C to 70°C, V_{CC} = 5V ±5%

Symbol	Parameter		Limits			Unit	Test Conditions
			Min	Typ(1)	Max		
V_C	Input Clamp Voltage, All Input			0.75	−1.0	V	V_{CC} = 4.75V; I_C = −5 mA
I_F	Input Load Current	$\overline{STSTB}$			500	µA	V_{CC} = 5.25V
		D_2 & D_6			750	µA	V_F = 0.45V
		D_0, D_1, D_4, D_5 & D_7			250	µA	
		All Other Inputs			250	µA	
I_R	Input Leakage Current	$\overline{STSTB}$			100	µA	V_{CC} = 5.25V
		DB_0–DB_7			20	µA	V_R = 5.25V
		All Other Inputs			100	µA	
V_{TH}	Input Threshold Voltage, All Inputs		0.8		2.0	V	V_{CC} = 5V
I_{CC}	Power Supply Current			140	190	mA	V_{CC} = 5.25V
V_{OL}	Output Low Voltage	D_0–D_7			0.45	V	V_{CC} = 4.75V; I_{OL} = 2 mA
		All Other Outputs			0.45	V	I_{OL} = 10 mA
V_{OH}	Output High Voltage	D_0–D_7	3.6	3.8		V	V_{CC} = 4.75V; I_{OH} = −10µA
		All Other Outputs	2.4			V	I_{OH} = −1 mA
I_{OS}	Short Circuit Current, All Outputs		15		90	mA	V_{CC} = 5V
$I_{O\ (off)}$	Off State Output Current All Control Outputs				100	µA	V_{CC} = 5.25V; V_O = 5.25V
					−100	µA	V_O = 0.45V
I_{INT}	INTA Current				5	mA	(See INTA Test Circuit)

NOTE:
1. Typical values are for T_A = 25°C and nominal supply voltages.

CAPACITANCE V_{BIAS} = 2.5V, V_{CC} = 5.0V, T_A = 25°C, f = 1 MHz

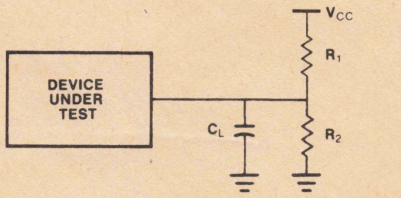
1. This parameter is periodically sampled and not 100% tested.

Symbol	Parameter	Limits			Unit
		Min	Typ(1)	Max	
C_{IN}	Input Capacitance		8	12	pF
C_{OUT}	Output Capacitance Control Signals		7	15	pF
I/O	I/O Capacitance (D or DB)		8	15	pF

A.C. CHARACTERISTICS T_A = 0°C to 70°C, V_{CC} = 5V ±5%

Symbol	Parameter	Limits		Unit	Conditions
		Min	Max		
t_{PW}	Width of Status Strobe	22		ns	
t_{SS}	Setup Time, Status Inputs D_0–D_7	8		ns	
t_{SH}	Hold Time, Status Inputs D_0–D_7	5		ns	
t_{DC}	Delay from $\overline{STSTB}$ to any Control Signal	20	60	ns	C_L = 100 pF
t_{RR}	Delay from DBIN to Control Outputs		30	ns	C_L = 100 pF
t_{RE}	Delay from DBIN to Enable/Disable 8080 Bus		45	ns	C_L = 25 pF
t_{RD}	Delay from System Bus to 8080 Bus during Read		30	ns	C_L = 25 pF
t_{WR}	Delay from $\overline{WR}$ to Control Outputs	5	45	ns	C_L = 100 pF
t_{WE}	Delay to Enable System Bus DB_0–DB_7 after $\overline{STSTB}$		30	ns	C_L = 100 pF
t_{WD}	Delay from 8080 Bus D_0–D_7 to System Bus DB_0–DB_7 during Write	5	40	ns	C_L = 100 pF
t_E	Delay from $\overline{System\ Bus\ Enable}$ to System Bus DB_0–DB_7		30	ns	C_L = 100 pF
t_{HD}	HLDA to Read Status Outputs		25	ns	
t_{DS}	Setup Time, System Bus Inputs to HLDA	10		ns	
t_{DH}	Hold Time, System Bus Inputs to HLDA	20		ns	C_L = 100 pF

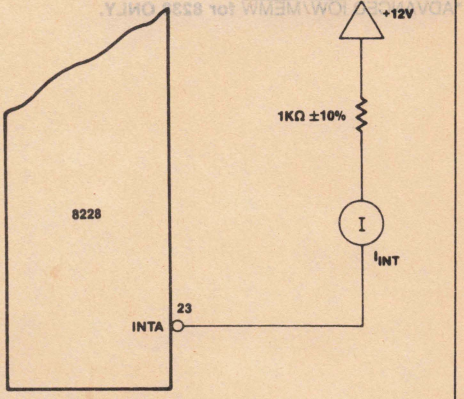
AC TESTING LOAD CIRCUIT



231465-3

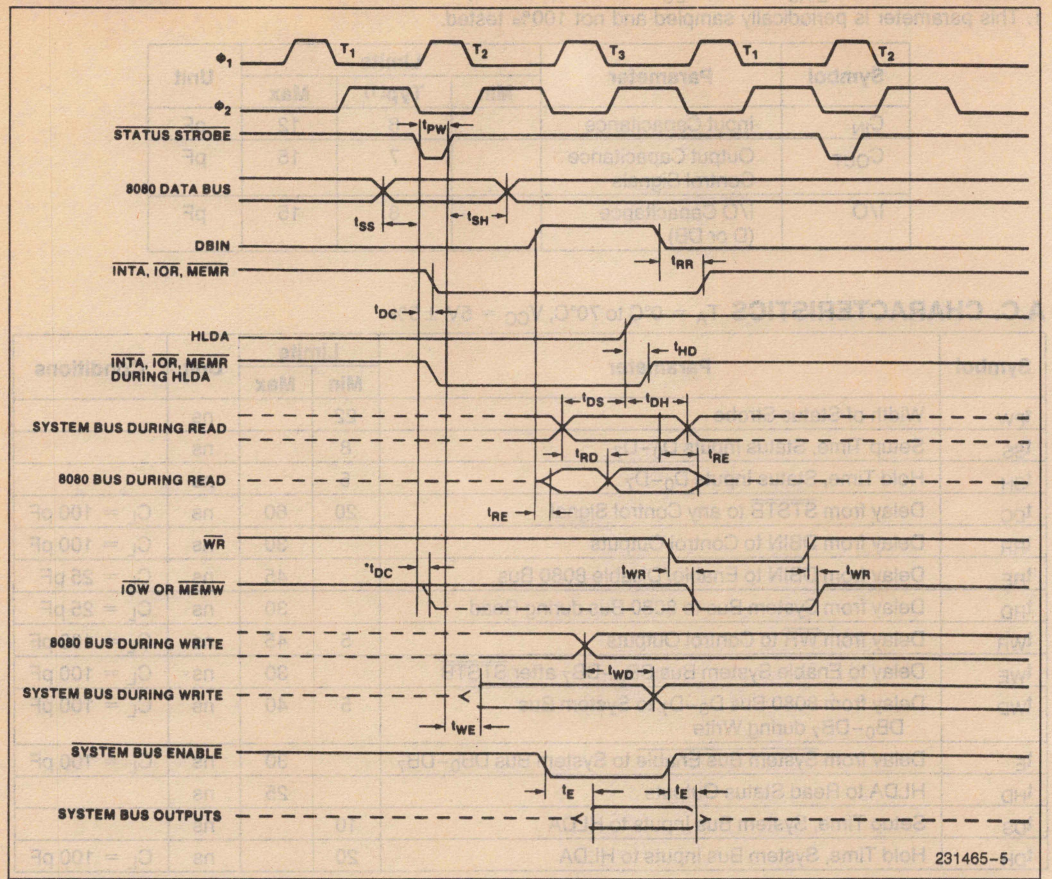
For D_0–D_7; R_1 = 4 KΩ, R_2 = ∞Ω, C_L = 25 pF.
For all other outputs: R_1 = 500Ω, R_2 = 1 KΩ, C_L = 100 pF.

INTA Test Circuit (for RST 7)



231465-4

WAVEFORMS



VOLTAGE MEASUREMENT POINTS: D_0–D_7 (when outputs) Logic "0" = 0.8V, Logic "1" = 3.0V. All other signals measured at 1.5V.

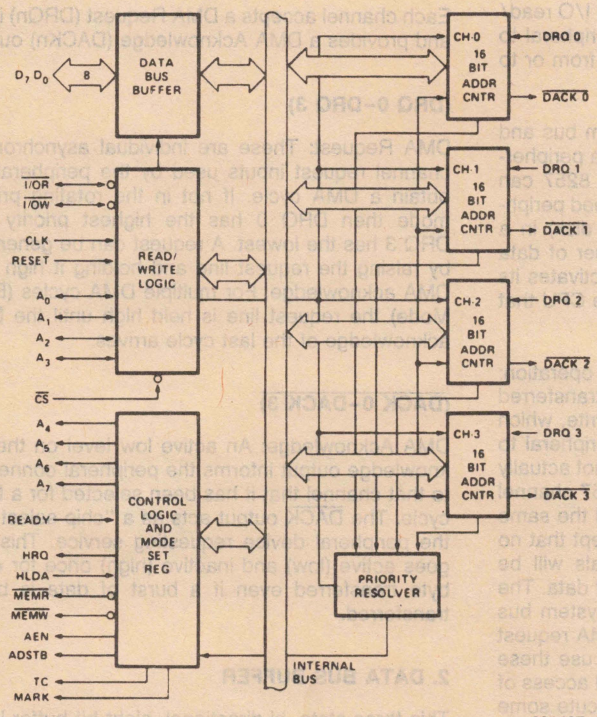
NOTE:
*ADVANCED $\overline{IOW}/\overline{MEMW}$ **for 8238 ONLY.**

8257/8257-5
PROGRAMMABLE DMA CONTROLLER

- MCS-85® Compatible 8257-5
- 4-Channel DMA Controller
- Priority DMA Request Logic
- Channel Inhibit Logic
- Terminal Count and Modulo 128 Outputs
- Single TTL Clock
- Single +5V Supply
- Auto Load Mode
- Available in EXPRESS — Standard Temperature Range
- Available in 40-Lead Cerdip and Plastic Package

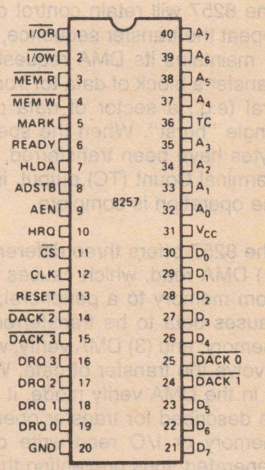
  (See Packaging Spec; Order #231369)

The Intel 8257 is a 4-channel direct memory access (DMA) controller. It is specifically designed to simplify the transfer of data at high speeds for the Intel microcomputer systems. Its primary function is to generate, upon a peripheral request, a sequential memory address which will allow the peripheral to read or write data directly to or from memory. Acquisition of the system bus is accomplished via the CPU's hold function. The 8257 has priority logic that resolves the peripherals requests and issues a composite hold request to the CPU. It maintains the DMA cycle count for each channel and outputs a control signal to notify the peripheral that the programmed number of DMA cycles is complete. Other output control signals simplify sectored data transfers. The 8257 represents a significant savings in component count for DMA-based microcomputer systems and greatly simplifies the transfer of data at high speed between peripherals and memories.



231467-1

Figure 1. Block Diagram



231467-2

Figure 2. Pin Configuration

November 1986
Order Number: 231467-001

FUNCTIONAL DESCRIPTION

General

The 8257 is a programmable, Direct Memory Access (DMA) device which, when coupled with a single 8-bit latch provides a complete four-channel DMA controller for use in Intel microcomputer systems. After being initialized by software, the 8257 can transfer a block of data, containing up to 16,384 bytes, between memory and a peripheral device directly, without further intervention required of the CPU. Upon receiving a DMA transfer request from an enabled peripheral, the 8257:

1. Acquires control of the system bus.
2. Acknowledges that requesting peripheral which is connected to the highest priority channel.
3. Outputs the least significant eight bits of the memory address onto system address lines A_0–A_7, outputs the most significant eight bits of the memory address to the 8-bit latch via the data bus (the outputs of the latch should drive address lines A_8–A_{15}), and
4. Generates the appropriate memory and I/O read/write control signals that cause the peripheral to receive or deposit a data byte directly from or to the addressed location in memory.

The 8257 will retain control of the system bus and repeat the transfer sequence, as long as a peripheral maintains its DMA request. Thus, the 8257 can transfer a block of data to/from a high speed peripheral (e.g., a sector of data on a floppy disk) in a single "burst". When the specified number of data bytes have been transferred, the 8257 activates its Terminal Count (TC) output, informing the CPU that the operation is complete.

The 8257 offers three different modes of operation: (1) DMA read, which causes data to be transferred from memory to a peripheral; (2) DMA write, which causes data to be transferred from a peripheral to memory; and (3) DMA verify, which does not actually involve the transfer of data. When an 8257 channel is in the DMA verify mode, it will respond the same as described for transfer operations, except that no memory or I/O read/write control signals will be generated, thus preventing the transfer of data. The 8257, however, will gain control of the system bus and will acknowledge the peripheral's DMA request for each DMA cycle. The peripheral can use these acknowledge signals to enable an internal access of each byte of a data block in order to execute some verification procedure, such as the accumulation of a CRC (Cyclic Redundancy Code) checkword. For example, a block of DMA verify cycles might follow a block of DMA read cycles (memory to peripheral) to allow the peripheral to verify its newly acquired data.

Block Diagram Description

1. DMA CHANNELS

The 8257 provides four separate DMA channels (labeled CH-0 to CH-3). Each channel includes two sixteen-bit registers: (1) a DMA address register, and (2) a terminal count register. Both registers must be initialized before a channel is enabled. The DMA address register is loaded with the address of the first memory location to be accessed. The value loaded into the low-order 14-bits of the terminal count register specifies the number of DMA cycles minus one before the Terminal Count (TC) output is activated. For instance, a terminal count of 0 would cause the TC output to be active in the first DMA cycle for that channel. In general, if N = the number of desired DMA cycles, load the value N − 1 into the low-order 14-bits of the terminal count register. The most significant two bits of the terminal count register specify the type of DMA operation for that channel.

These two bits are not modified during a DMA cycle, but can be changed between DMA blocks.

Each channel accepts a DMA Request (DRQn) input and provides a DMA Acknowledge (DACKn) output.

(DRQ 0–DRQ 3)

DMA Request: These are individual asynchronous channel request inputs used by the peripherals to obtain a DMA cycle. If not in the rotating priority mode then DRQ 0 has the highest priority and DRQ 3 has the lowest. A request can be generated by raising the request line and holding it high until DMA acknowledge. For multiple DMA cycles (Burst Mode) the request line is held high until the DMA acknowledge of the last cycle arrives.

($\overline{\text{DACK}}$ 0–$\overline{\text{DACK}}$ 3)

DMA Acknowledge: An active low level on the acknowledge output informs the peripheral connected to that channel that it has been selected for a DMA cycle. The $\overline{\text{DACK}}$ output acts as a "chip select" for the peripheral device requesting service. This line goes active (low) and inactive (high) once for each byte transferred even if a burst of data is being transferred.

2. DATA BUS BUFFER

This three-state, bi-directional, eight bit buffer interfaces the 8257 to the system data bus.

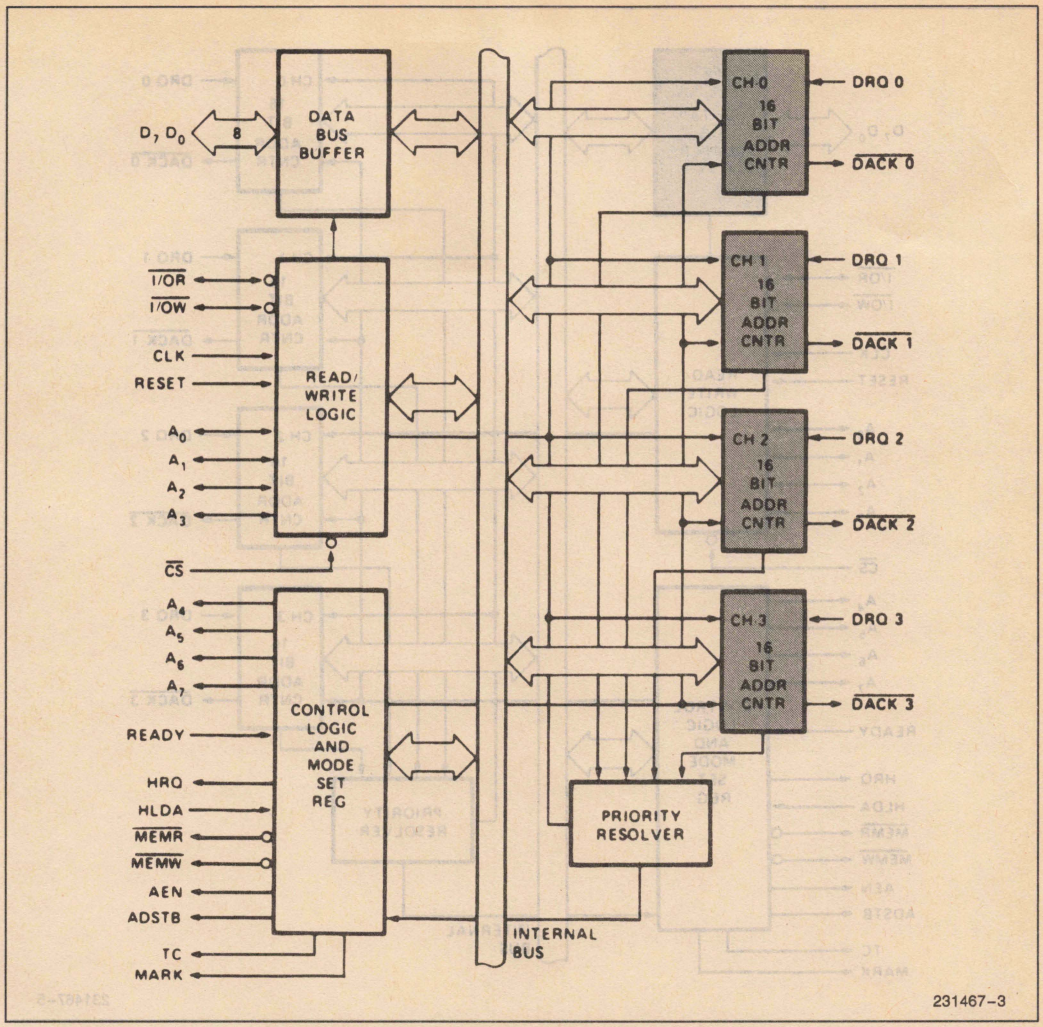


Figure 3. 8257 Block Diagram Showing DMA Channels

(D_0–D_7)

Data Bus Lines: These are bi-directional three-state lines. When the 8257 is being programmed by the CPU, eight-bits of data for a DMA address register, a terminal count register or the Mode Set register are received on the data bus. When the CPU reads a DMA address register, a terminal count register or the Status register, the data is sent to the CPU over the data bus. During DMA cycles (when the 8257 is the bus master), the 8257 will output the most significant eight-bits of the memory address (from one of the DMA address registers) to the 8212 latch via the data bus. These address bits will be transferred at the beginning of the DMA cycle; the bus will then be released to handle the memory data transfer during the balance of the DMA cycle.

Bit 15	Bit 14	Type of DMA Operation
0	0	Verify DMA Cycle
0	1	Write DMA Cycle
1	0	Read DMA Cycle
1	1	(Illegal)

3. Read/Write Logic

When the CPU is programming or reading one of the 8257's registers (i.e., when the 8257 is a "slave"

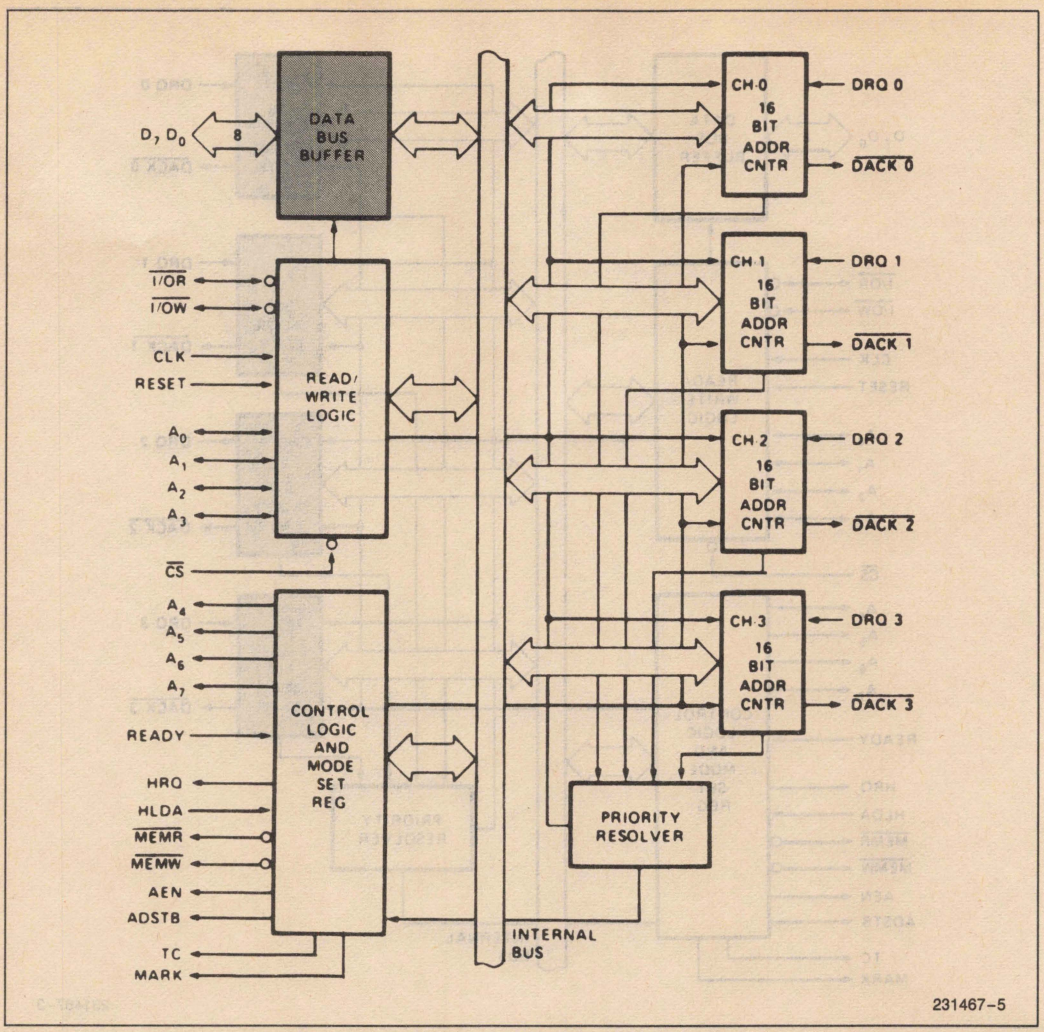


Figure 4. 8257 Block Diagram Showing Data Bus Buffer

device on the system bus), the Read/Write Logic accepts the I/O Read ($\overline{I/OR}$) or I/O Write ($\overline{I/OW}$) signal, decodes the least significant four address bits, (A_0–A_3), and either writes the contents of the data bus into the addressed register (if $\overline{I/OW}$ is true) or places the contents of the addressed register onto the data bus (if $\overline{I/OR}$ is true).

During DMA cycles (i.e., when the 8257 is the bus "master"), the Read/Write Logic generates the I/O read and memory write (DMA write cycle) or I/O Write and memory read (DMA read cycle) signals which control the data link with the peripheral that has been granted the DMA cycle.

Note that during DMA transfers Non-DMA I/O devices should be de-selected (disabled) using "AEN" signal to inhibit I/O device decoding of the memory address as an erroneous device address.

$\overline{(I/OR)}$

I/O Read: An active-low, bi-directional three-state line. In the "slave" mode, it is an input which allows the 8-bit status register or the upper/lower byte of a 16-bit DMA address register or terminal count register to be read. In the "master" mode, $\overline{I/OR}$ is a control output which is used to access data from a peripheral during the DMA write cycle.

($\overline{I/OW}$)

I/O Write: An active-low, bi-directional three-state line. In the "slave" mode, it is an input which allows the contents of the data bus to be loaded into the 8-bit mode set register or the upper/lower byte of a 16-bit DMA address register or terminal count register. In the "master" mode, $\overline{I/OW}$ is a control output which allows data to be output to a peripheral during a DMA read cycle.

(CLK)

Clock Input: Generally from an Intel 8224 Clock Generator device. (ϕ2 TTL) or Intel 8085AH CLK output.

(RESET)

Reset: An asynchronous input (generally from an 8224 or 8085 device) which disables all DMA channels by clearing the mode register and 3-states all control lines.

(A_0–A_3)

Address Lines: These least significant four address lines are bi-directional. In the "slave" mode they are inputs which select one of the registers to be read or programmed. In the "master" mode, they are outputs which constitute the least significant four bits of the 16-bit memory address generated by the 8257.

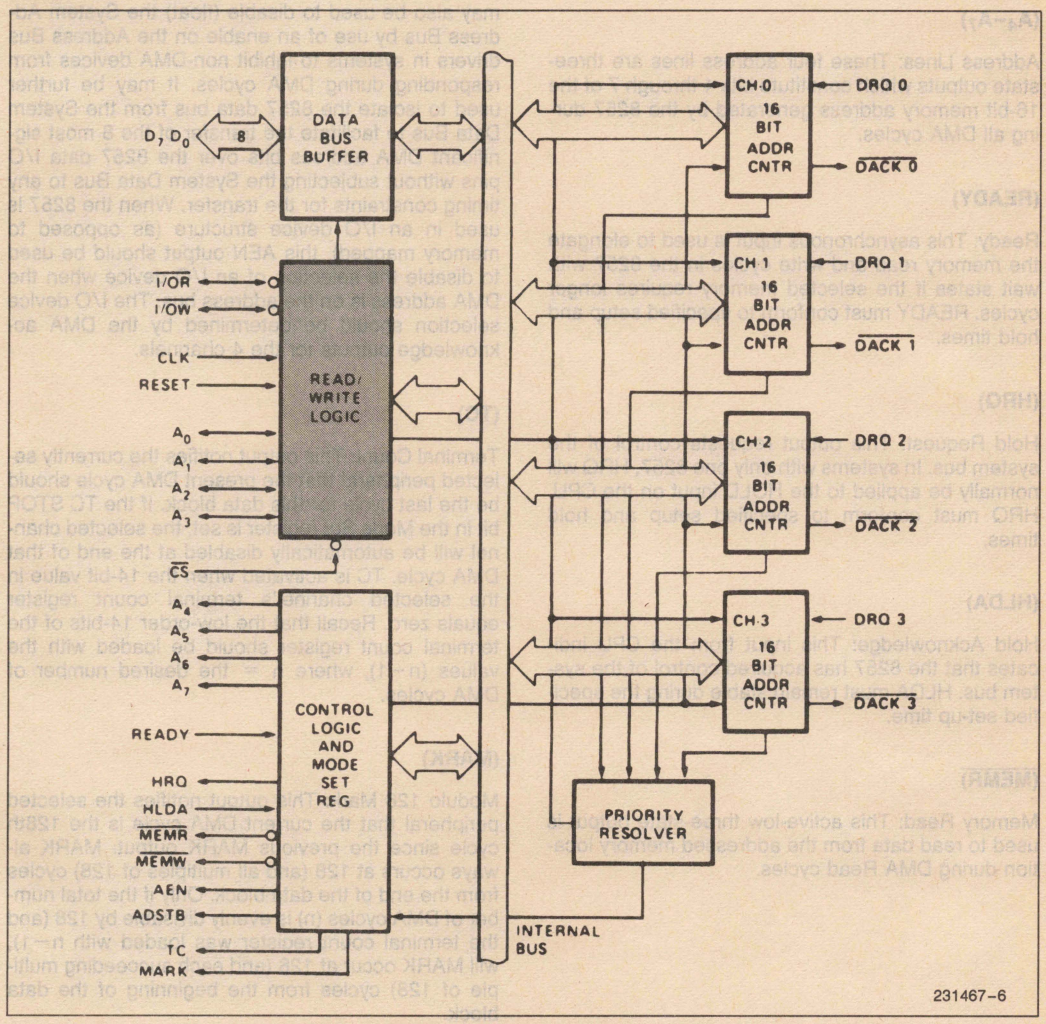


Figure 5. 8257 Block Diagram Showing Read/Write Logic Function

$\overline{(CS)}$

Chip Select: An active-low input which enables the I/O Read or I/O Write input when the 8257 is being read or programmed in the "slave" mode. In the "master" mode, $\overline{CS}$ is automatically disabled to prevent the chip from selecting itself while performing the DMA function.

4. CONTROL LOGIC

This block controls the sequence of operations during all DMA cycles by generating the appropriate control signals and the 16-bit address that specifies the memory location to be accessed.

(A_4-A_7)

Address Lines: These four address lines are three-state outputs which constitute bits 4 through 7 of the 16-bit memory address generated by the 8257 during all DMA cycles.

(READY)

Ready: This asynchronous input is used to elongate the memory read and write cycles in the 8257 with wait states if the selected memory requires longer cycles. READY must conform to specified setup and hold times.

(HRQ)

Hold Request: This output requests control of the system bus. In systems with only one 8257, HRQ will normally be applied to the HOLD input on the CPU. HRQ must conform to specified setup and hold times.

(HLDA)

Hold Acknowledge: This input from the CPU indicates that the 8257 has acquired control of the system bus. HLDA must remain stable during the specified set-up time.

$\overline{(MEMR)}$

Memory Read: This active-low three-state output is used to read data from the addressed memory location during DMA Read cycles.

$\overline{(MEMW)}$

Memory Write: This active-low three-state output is used to write data into the addressed memory location during DMA Write cycles.

(ADSTB)

Address Strobe: This output strobes the most significant byte of the memory address into the latch device from the data bus.

(AEN)

Address Enable: This output is used to disable (float) the System Data Bus and the System Control Bus. It may also be used to disable (float) the System Address Bus by use of an enable on the Address Bus drivers in systems to inhibit non-DMA devices from responding during DMA cycles. It may be further used to isolate the 8257 data bus from the System Data Bus to facilitate the transfer of the 8 most significant DMA address bits over the 8257 data I/O pins without subjecting the System Data Bus to any timing constraints for the transfer. When the 8257 is used in an I/O device structure (as opposed to memory mapped), this AEN output should be used to disable the selection of an I/O device when the DMA address is on the address bus. The I/O device selection should be determined by the DMA acknowledge outputs for the 4 channels.

(TC)

Terminal Count: This output notifies the currently selected peripheral that the present DMA cycle should be the last cycle for this data block. If the TC STOP bit in the Mode Set register is set, the selected channel will be automatically disabled at the end of that DMA cycle. TC is activated when the 14-bit value in the selected channel's terminal count register equals zero. Recall that the low-order 14-bits of the terminal count register should be loaded with the values (n−1), where n = the desired number of DMA cycles.

(MARK)

Modulo 128 Mark: This output notifies the selected peripheral that the current DMA cycle is the 128th cycle since the previous MARK output. MARK always occurs at 128 (and all multiples of 128) cycles from the end of the data block. Only if the total number of DMA cycles (n) is evenly divisable by 128 (and the terminal count register was loaded with n−1), will MARK occur at 128 (and each succeeding multiple of 128) cycles from the beginning of the data block.

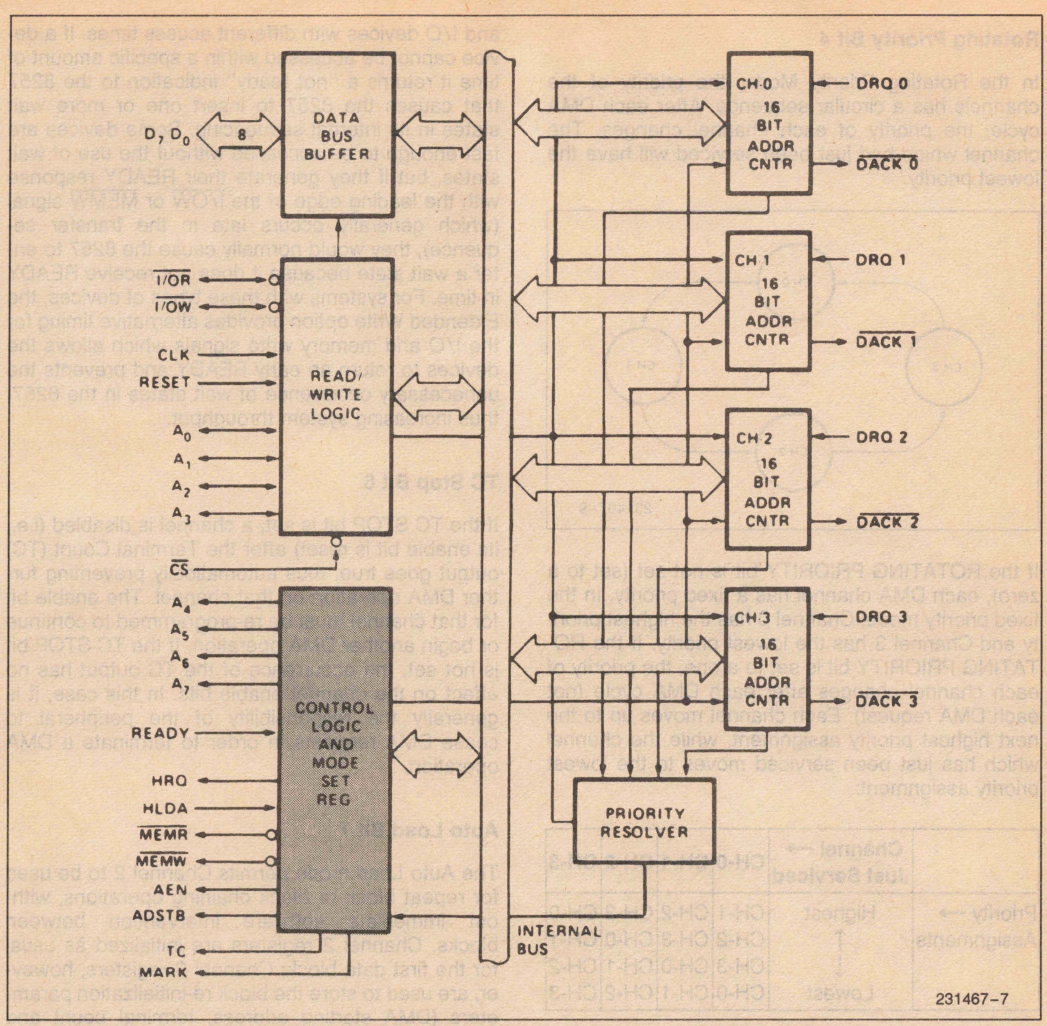
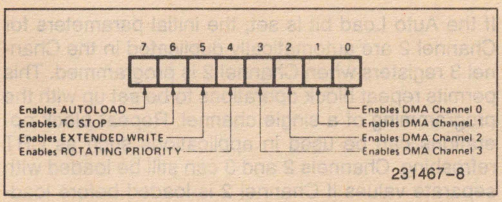


Figure 6. 8257 Block Diagram Showing Control Logic and Mode Set Register

5. MODE SET REGISTER

When set, the various bits in the Mode Set register enable each of the four DMA channels, and allow four different options for the 8257:

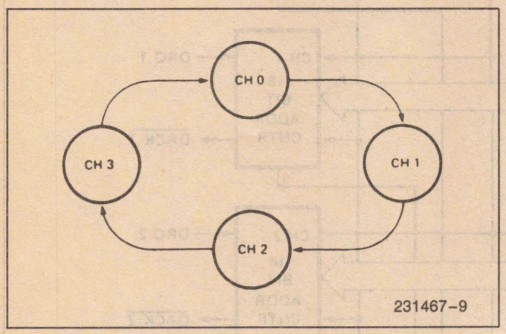


The Mode Set register is normally programmed by the CPU after the DMA address register(s) and terminal count register(s) are initialized. The Mode Set Register is cleared by the RESET input, thus disabling all options, inhibiting all channels, and preventing bus conflicts on power-up. A channel should not be left enabled unless its DMA address and terminal count registers contain valid values; otherwise, an inadvertent DMA request (DRQn) from a peripheral could initiate a DMA cycle that would destroy memory data.

The various options which can be enabled by bits in the Mode Set register are explained below.

Rotating Priority Bit 4

In the Rotating Priority Mode, the priority of the channels has a circular sequence. After each DMA cycle, the priority of each channel changes. The channel which had just been serviced will have the lowest priority.



If the ROTATING PRIORITY bit is not set (set to a zero), each DMA channel has a fixed priority. In the fixed priority mode, Channel 0 has the highest priority and Channel 3 has the lowest priority. If the ROTATING PRIORITY bit is set to a one, the priority of each channel changes after each DMA cycle (not each DMA request). Each channel moves up to the next highest priority assignment, while the channel which has just been serviced moves to the lowest priority assignment:

	Channel → Just Serviced	CH-0	CH-1	CH-2	CH-3
Priority → Assignments	Highest	CH-1	CH-2	CH-3	CH-0
	↑	CH-2	CH-3	CH-0	CH-1
	↓	CH-3	CH-0	CH-1	CH-2
	Lowest	CH-0	CH-1	CH-2	CH-3

Note that rotating priority will prevent any one channel from monopolizing the DMA mode; consecutive DMA cycles will service different channels if more than one channel is enabled and requesting service. There is no overhead penalty associated with this mode of operation. All DMA operations began with Channel 0 initially assigned to the highest priority for the first DMA cycle.

Extended Write Bit 5

If the EXTENDED WRITE bit is set, the duration of both the $\overline{\text{MEMW}}$ and $\overline{\text{I/OW}}$ signals is extended by activating them earlier in the DMA cycle. Data transfers within microcomputer systems proceed asynchronously to allow use of various types of memory and I/O devices with different access times. If a device cannot be accessed within a specific amount of time it returns a "not ready" indication to the 8257 that causes the 8257 to insert one or more wait states in its internal sequencing. Some devices are fast enough to be accessed without the use of wait states, but if they generate their READY response with the leading edge of the $\overline{\text{I/OW}}$ or $\overline{\text{MEMW}}$ signal (which generally occurs late in the transfer sequence), they would normally cause the 8257 to enter a wait state because it does not receive READY in time. For systems with these types of devices, the Extended Write option provides alternative timing for the I/O and memory write signals which allows the devices to return an early READY and prevents the unnecessary occurrence of wait states in the 8257, thus increasing system throughput.

TC Stop Bit 6

If the TC STOP bit is set, a channel is disabled (i.e., its enable bit is reset) after the Terminal Count (TC) output goes true, thus automatically preventing further DMA operation on that channel. The enable bit for that channel must be re-programmed to continue or begin another DMA operation. If the TC STOP bit is not set, the occurrence of the TC output has no effect on the channel enable bits. In this case, it is generally the responsibility of the peripheral to cease DMA requests in order to terminate a DMA operation.

Auto Load Bit 7

The Auto Load mode permits Channel 2 to be used for repeat block or block chaining operations, without immediate software intervention between blocks. Channel 2 registers are initialized as usual for the first data block; Channel 3 registers, however, are used to store the block re-initialization parameters (DMA starting address, terminal count and DMA transfer mode). After the first block of DMA cycles is executed by Channel 2 (i.e., after the TC output goes true), the parameters stored in the Channel 3 registers are transferred to Channel 2 during an "update" cycle. Note that the TC STOP feature, described above, has no effect on Channel 2 when the Auto Load bit is set.

If the Auto Load bit is set, the initial parameters for Channel 2 are automatically duplicated in the Channel 3 registers when Channel 2 is programmed. This permits repeat block operations to be set up with the programming of a single channel. Repeat block operations can be used in applications such as CRT refreshing. Channels 2 and 3 can still be loaded with separate values if Channel 2 is loaded before loading Channel 3. Note that in the Auto Load

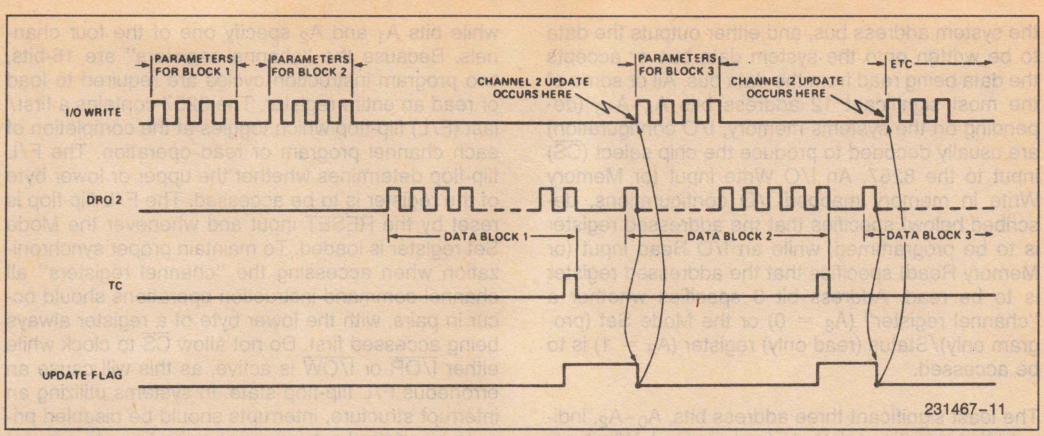


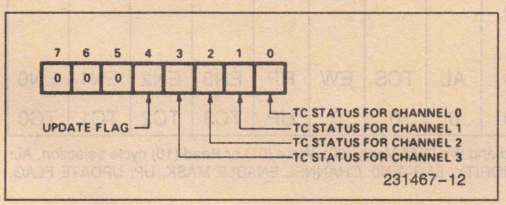
Figure 7. Autoload Timing

mode, Channel 3 is still available to the user if the Channel 3 enable bit is set, but use of this channel will change the values to be auto loaded into Channel 2 at update time. All that is necessary to use the Auto Load feature for chaining operations is to reload Channel 3 registers at the conclusion of each update cycle with the new parameters for the next data block transfer.

Each time that the 8257 enters an update cycle, the update flag in the status register is set and parameters in Channel 3 are transferred to Channel 2, nondestructively for Channel 3. The actual re-initialization of Channel 2 occurs at the beginning of the next Channel 2 DMA cycle after the TC cycle. This will be the first DMA cycle of the new data block for Channel 2. The update flag is cleared at the conclusion of this DMA cycle. For chaining operations, the update flag in the status register can be monitored by the CPU to determine when the re-initialization process has been completed so that the next block parameters can be safely loaded into Channel 3.

6. Status Register

The eight-bit status register indicates which channels have reached a terminal count condition and includes the update flag described previously.



The TC status bits are set when the Terminal Count (TC) output is activated for that channel. These bits remain set until the status register is read or the 8257 is reset. The UPDATE FLAG, however, is not affected by a status register read operation. The UPDATE FLAG can be cleared by resetting the 8257, by changing to the non-auto load mode (i.e., by resetting the AUTO LOAD bit in the Mode Set register) or it can be left to clear itself at the completion of the update cycle. The purpose of the UPDATE FLAG is to prevent the CPU from inadvertently skipping a data block by overwriting a starting address or terminal count in the Channel 3 registers before those parameters are properly auto-loaded into Channel 2.

The user is cautioned against reading the TC status register and using this information to reenable channels that have not completed operation. Unless the DMA channels are inhibited a channel could reach terminal count (TC) between the status read and the mode write. DMA can be inhibited by a hardware gate on the HRQ line or by disabling channels with a mode word before reading the TC status.

OPERATIONAL SUMMARY

Programming and Reading the 8257 Registers

There are four pairs of "channel registers": each pair consisting of a 16-bit DMA address register and a 16-bit terminal count register (one pair for each channel). The 8257 also includes two "general registers": one 8-bit Mode Set register and one 8-bit Status register. The registers are loaded or read when the CPU executes a write or read instruction that addresses the 8257 device and the appropriate register within the 8257. The 8228 generates the appropriate read or write control signal (generally I/OR or I/OW while the CPU places a 16-bit address on

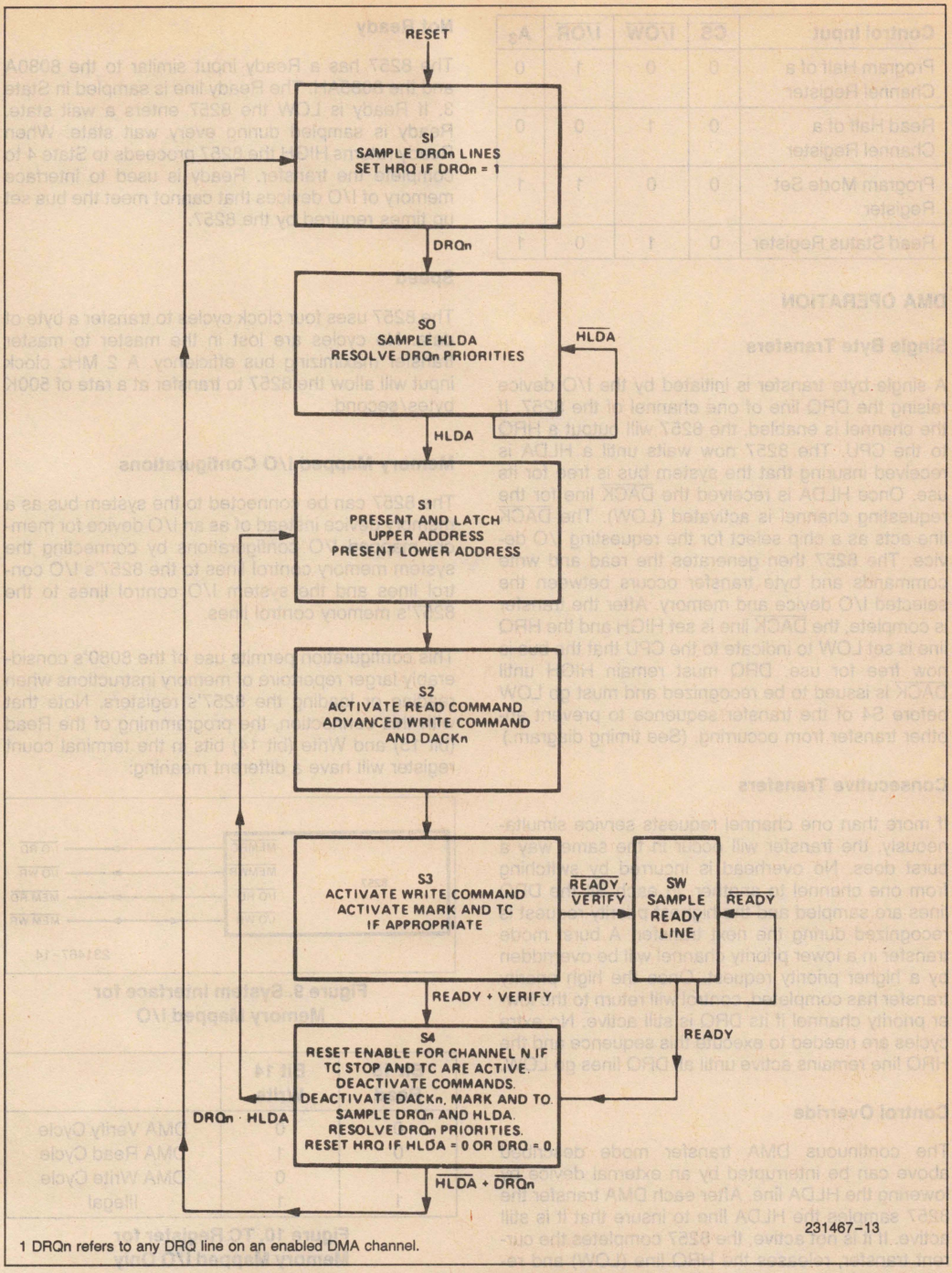
the system address bus, and either outputs the data to be written onto the system data bus or accepts the data being read from the data bus. All or some of the most significant 12 address bits A_4–A_{15} (depending on the systems memory, I/O configuration) are usually decoded to produce the chip select ($\overline{CS}$) input to the 8257. An I/O Write input (or Memory Write in memory mapped I/O configurations, described below) specifies that the addressed register is to be programmed, while an I/O Read input (or Memory Read) specifies that the addressed register is to be read. Address bit 3 specifies whether a "channel register" ($A_3 = 0$) or the Mode Set (program only)/Status (read only) register ($A_3 = 1$) is to be accessed.

The least significant three address bits, A_0–A_2, indicate the specific register to be accessed. When accessing the Mode Set or Status register, A_0–A_2 are all zero. When accessing a channel register bit A_0 differentiates between the DMA address register ($A_0 = 0$) and the terminal count register ($A_0 = 1$), while bits A_1 and A_2 specify one of the four channels. Because the "channel registers" are 16-bits, two program instruction cycles are required to load or read an entire register. The 8257 contains a first/last (F/L) flip-flop which toggles at the completion of each channel program or read operation. The F/L flip-flop determines whether the upper or lower byte of the register is to be accessed. The F/L flip-flop is reset by the RESET input and whenever the Mode Set register is loaded. To maintain proper synchronization when accessing the "channel registers" all channel command instruction operations should occur in pairs, with the lower byte of a register always being accessed first. Do not allow $\overline{CS}$ to clock while either $\overline{I/OR}$ or $\overline{I/OW}$ is active, as this will cause an erroneous F/L flip-flop state. In systems utilizing an interrupt structure, interrupts should be disabled prior to any paired programming operations to prevent an interrupt from splitting them. The result of such a split would leave the F/L F/F in the wrong state. This problem is particularly obvious when other DMA channels are programmed by an interrupt structure.

8257 Register Selection

Register	Byte	Address Inputs				F/L	*Bi-Directional Data Bus							
		A_3	A_2	A_1	A_0		D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
CH-0 DMA Address	LSB	0	0	0	0	0	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
	MSB	0	0	0	0	1	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8
CH-0 Terminal Count	LSB	0	0	0	1	0	C_7	C_6	C_5	C_4	C_3	C_2	C_1	C_0
	MSB	0	0	0	1	1	Rd	Wr	C_{13}	C_{12}	C_{11}	C_{10}	C_9	C_8
CH-1 DMA Address	LSB	0	0	1	0	0	Same as Channel 0							
	MSB	0	0	1	0	1								
CH-1 Terminal Count	LSB	0	0	1	1	0								
	MSB	0	0	1	1	1								
CH-2 DMA Address	LSB	0	1	0	0	0	Same as Channel 0							
	MSB	0	1	0	0	1								
CH-2 Terminal Count	LSB	0	1	0	1	0								
	MSB	0	1	0	1	1								
CH-3 DMA Address	LSB	0	1	1	0	0	Same as Channel 0							
	MSB	0	1	1	0	1								
CH-3 Terminal Count	LSB	0	1	1	1	0								
	MSB	0	1	1	1	1								
Mode Set (Program only)	—	1	0	0	0	0	AL	TCS	EW	RP	EN3	EN2	EN1	EN0
Status (Read only)	—	1	0	0	0	0	0	0	0	UP	TC3	TC2	TC1	TC0

*A_0–A_{15}: DMA Starting Address, C_0–C_{13}: Terminal Count value (N − 1), Rd and Wr: DMA Verify (00), Write (01) or Read (10) cycle selection, AL: Auto Load, TCS: TC STOP, EW: EXTENDED WRITE, RP: ROTATING PRIORITY, EN3–EN0: CHANNEL ENABLE MASK, UP: UPDATE FLAG, TC3–TC0: TERMINAL COUNT STATUS BITS.



1 DRQn refers to any DRQ line on an enabled DMA channel.

Figure 8. DMA Operation State Diagram

Control Input	$\overline{CS}$	$\overline{I/OW}$	$\overline{I/OR}$	A_3
Program Half of a Channel Register	0	0	1	0
Read Half of a Channel Register	0	1	0	0
Program Mode Set Register	0	0	1	1
Read Status Register	0	1	0	1

DMA OPERATION

Single Byte Transfers

A single byte transfer is initiated by the I/O device raising the DRQ line of one channel of the 8257. If the channel is enabled, the 8257 will output a HRQ to the CPU. The 8257 now waits until a HLDA is received insuring that the system bus is free for its use. Once HLDA is received the $\overline{DACK}$ line for the requesting channel is activated (LOW). The $\overline{DACK}$ line acts as a chip select for the requesting I/O device. The 8257 then generates the read and write commands and byte transfer occurs between the selected I/O device and memory. After the transfer is complete, the $\overline{DACK}$ line is set HIGH and the HRQ line is set LOW to indicate to the CPU that the bus is now free for use. DRQ must remain HIGH until $\overline{DACK}$ is issued to be recognized and must go LOW before S4 of the transfer sequence to prevent another transfer from occurring. (See timing diagram.)

Consecutive Transfers

If more than one channel requests service simultaneously, the transfer will occur in the same way a burst does. No overhead is incurred by switching from one channel to another. In each S4 the DRQ lines are sampled and the highest priority request is recognized during the next transfer. A burst mode transfer in a lower priority channel will be overridden by a higher priority request. Once the high priority transfer has completed, control will return to the lower priority channel if its DRQ is still active. No extra cycles are needed to execute this sequence and the HRQ line remains active until all DRQ lines go LOW.

Control Override

The continuous DMA transfer mode described above can be interrupted by an external device by lowering the HLDA line. After each DMA transfer the 8257 samples the HLDA line to insure that it is still active. If it is not active, the 8257 completes the current transfer, releases the HRQ line (LOW) and returns to the idle state. If DRQ lines are still active the 8257 will raise the HRQ line in the third cycle and proceed normally. (See timing diagram.)

Not Ready

The 8257 has a Ready input similar to the 8080A and the 8085AH. The Ready line is sampled in State 3. If Ready is LOW the 8257 enters a wait state. Ready is sampled during every wait state. When Ready returns HIGH the 8257 proceeds to State 4 to complete the transfer. Ready is used to interface memory of I/O devices that cannot meet the bus set up times required by the 8257.

Speed

The 8257 uses four clock cycles to transfer a byte of data. No cycles are lost in the master to master transfer maximizing bus efficiency. A 2 MHz clock input will allow the 8257 to transfer at a rate of 500K bytes/second.

Memory Mapped I/O Configurations

The 8257 can be connected to the system bus as a memory device instead of as an I/O device for memory mapped I/O configurations by connecting the system memory control lines to the 8257's I/O control lines and the system I/O control lines to the 8257's memory control lines.

This configuration permits use of the 8080's considerably larger repertoire of memory instructions when reading or loading the 8257's registers. Note that with this connection, the programming of the Read (bit 15) and Write (bit 14) bits in the terminal count register will have a different meaning:

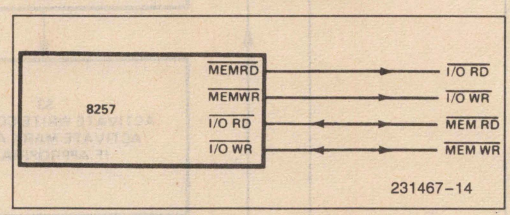


Figure 9. System Interface for Memory Mapped I/O

Bit 15 Read	Bit 14 Write	
0	0	DMA Verify Cycle
0	1	DMA Read Cycle
1	0	DMA Write Cycle
1	1	Illegal

Figure 10. TC Register for Memory Mapped I/O Only

SYSTEM APPLICATION EXAMPLES

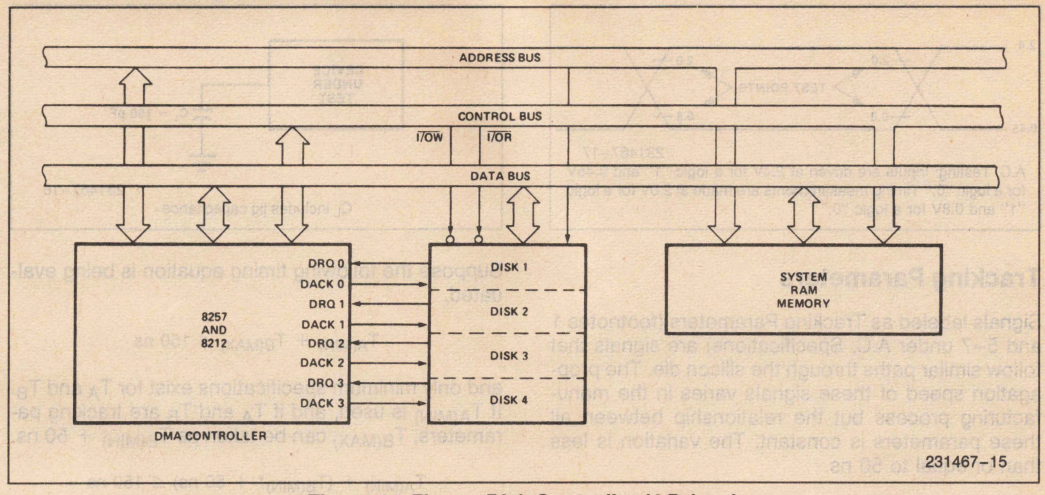


Figure 11. Floppy Disk Controller (4 Drives)

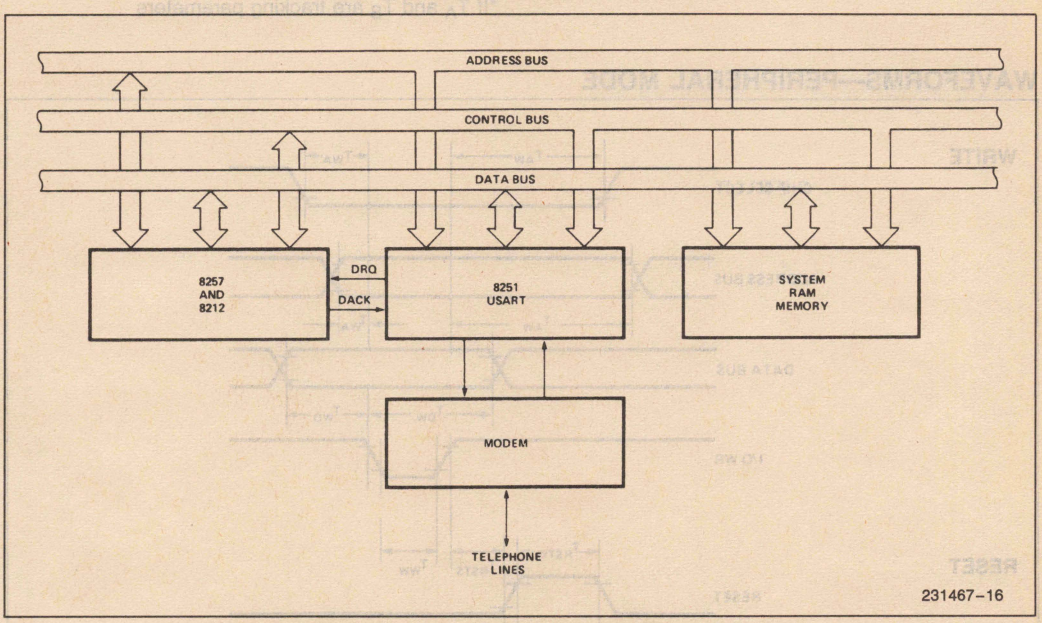
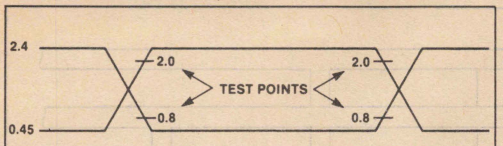


Figure 12. High-Speed Communication Controller

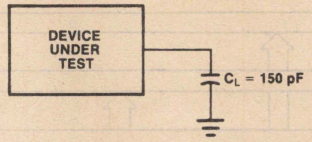
A.C. TESTING INPUT, OUTPUT WAVEFORM



231467-17

A.C. Testing: Inputs are driven at 2.4V for a logic "1" and 0.45V for a logic "0." Timing measurements are made at 2.0V for a logic "1" and 0.8V for a logic "0."

A.C. TESTING LOAD CIRCUIT



231467-18

C_L includes jig capacitance

Tracking Parameters

Signals labeled as Tracking Parameters (footnotes 1 and 5–7 under A.C. Specifications) are signals that follow similar paths through the silicon die. The propagation speed of these signals varies in the manufacturing process but the relationship between all these parameters is constant. The variation is less than or equal to 50 ns.

Suppose the following timing equation is being evaluated,

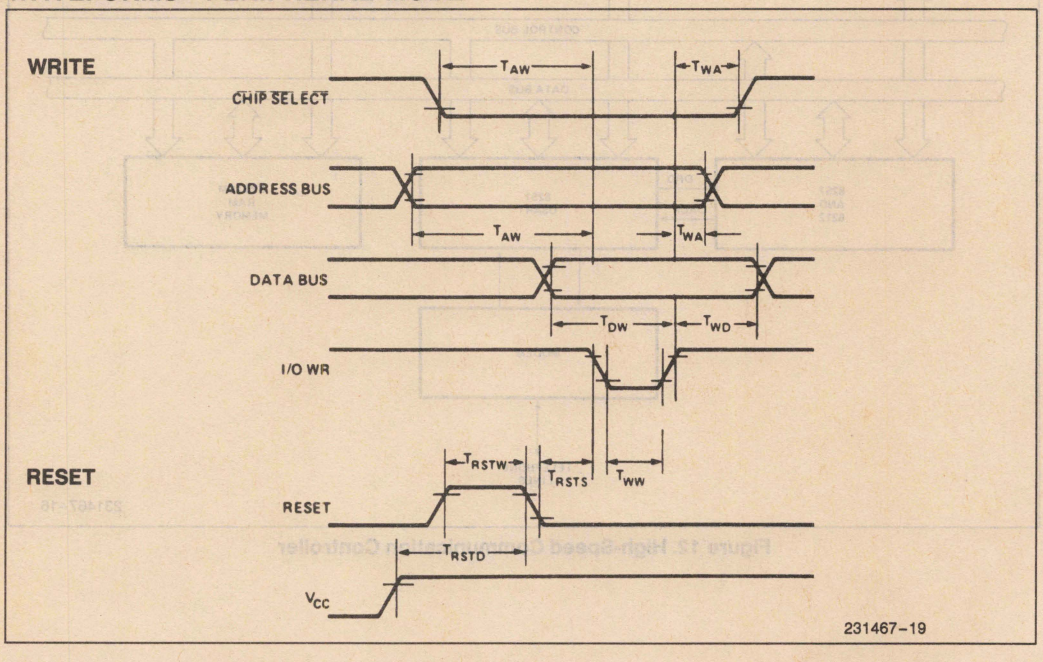
$$T_{A(MIN)} + T_{B(MAX)} \leq 150 \text{ ns}$$

and only minimum specifications exist for T_A and T_B. If $T_{A(MIN)}$ is used, and if T_A and T_B are tracking parameters, $T_{B(MAX)}$ can be taken as $T_{B(MIN)} + 50$ ns.

$$T_{A(MIN)} + (T_{B(MIN)}^* + 50 \text{ ns}) \leq 150 \text{ ns}$$

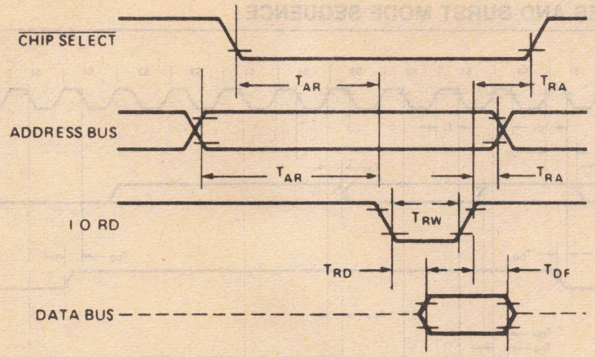
*If T_A and T_B are tracking parameters

WAVEFORMS—PERIPHERAL MODE



231467-19

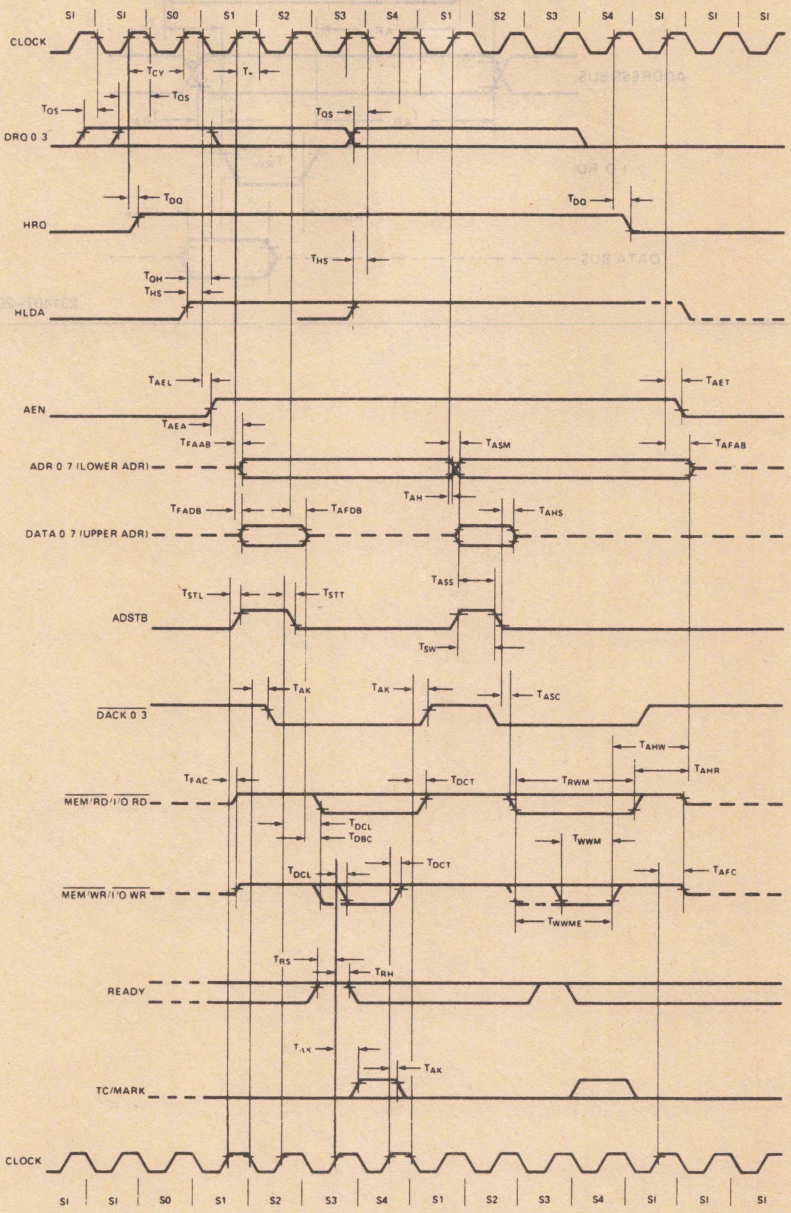
READ



231467-20

WAVEFORMS—DMA

CONSECUTIVE CYCLES AND BURST MODE SEQUENCE

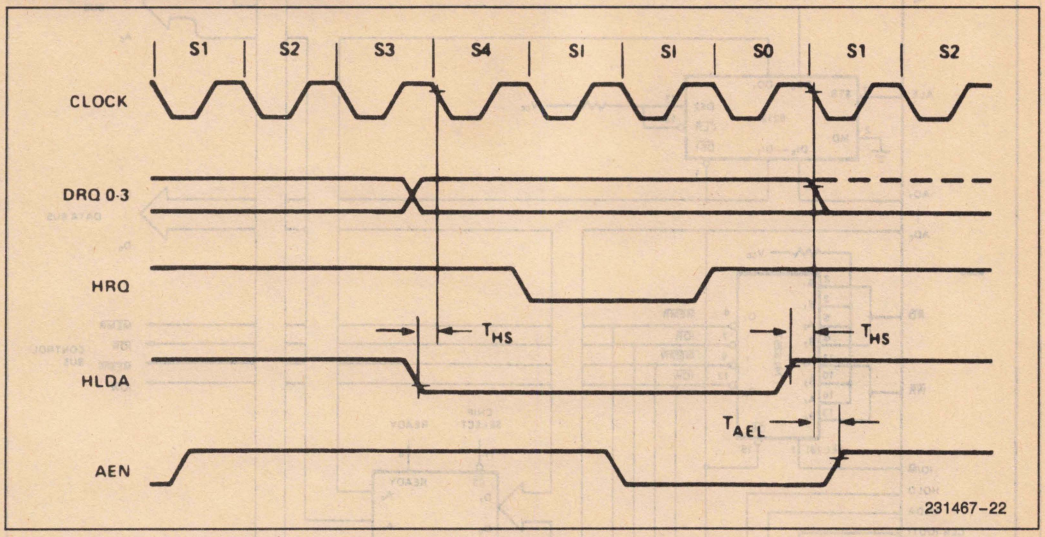


231467-21

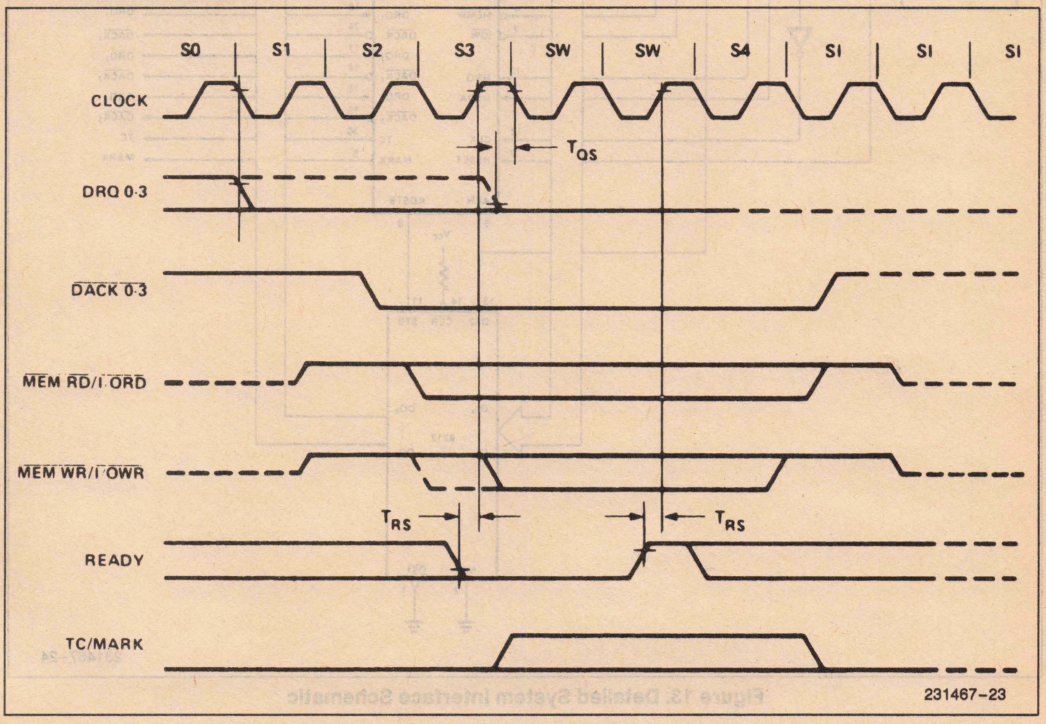
NOTE:
The clock waveform is duplicated for clarity. The 8257 requires only one clock input.

WAVEFORMS (Continued)

CONTROL OVERRIDE SEQUENCE



NOT READY SEQUENCE



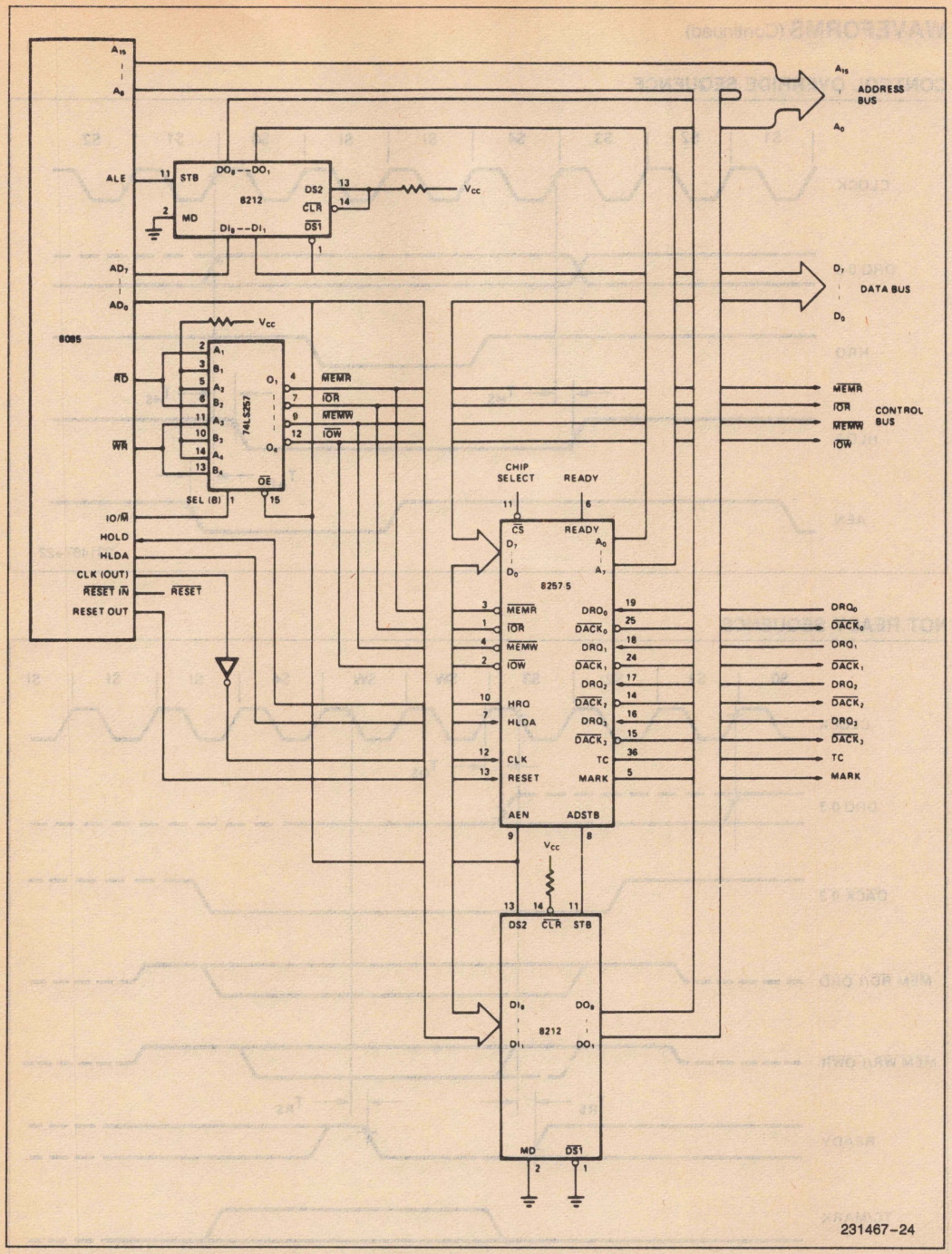


Figure 13. Detailed System Interface Schematic

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias .....0°C to 70°C
Storage Temperature .....−65°C to +150°C
Voltage on Any Pin
with Respect to Ground.....−0.5V to +7V
Power Dissipation .....1 Watt

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS

8257: T_A = 0°C to 70°C, V_{CC} = 5.0V ±5%, GND = 0V
8257-5: T_A = 0°C to 70°C, V_{CC} = 5.0V ±10%, GND = 0V

Symbol	Parameter	Min	Max	Unit	Test Conditions
V_{IL}	Input Low Voltage	−0.5	0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC}+0.5	V	
V_{OL}	Output Low Voltage		0.45	V	I_{OL} = 1.6 mA
V_{OH}	Output High Voltage	2.4	V_{CC}	V	I_{OH} = −150 μA for AB, DB and AEN I_{OH} = −80 μA for others
V_{HH}	HRQ Output High Voltage	3.3	V_{CC}	V	I_{OH} = −80 μA
I_{CC}	V_{CC} Current Drain		120	mA	
I_L	Input Leakage		±10	μA	0V ≤ V_{IN} ≤ V_{CC}
I_{OFL}	Output Leakage During Float		±10	μA	0.45V ≤ V_{OUT} ≤ V_{CC}

CAPACITANCE

T_A = 25°C; V_{CC} = GND = 0V

Symbol	Parameter	Min	Typ	Max	Unit	Test Conditions
C_{IN}	Input Capacitance			10	pF	fc = 1 MHz
$C_{I/O}$	I/O Capacitance			20	pF	Unmeasured pins returned to GND

A.C. CHARACTERISTICS—PERIPHERAL (SLAVE) MODE

8257: T_A = 0°C to 70°C, V_{CC} = 5.0V ±5%, GND = 0V
8257-5: T_A = 0°C to 70°C, V_{CC} = 5.0V ±10%, GND = 0V

8080 Bus Parameters

READ CYCLE

Symbol	Parameter	8257		8257-5		Unit	Test Conditions
		Min	Max	Min	Max		
T_{AR}	Adr or $\overline{CS}$ ↓ Setup to $\overline{RD}$ ↓	0		0		ns	
T_{RA}	Adr or $\overline{CS}$ ↑ Hold from $\overline{RD}$ ↑	0		0		ns	
T_{RD}	Data Access from $\overline{RD}$ ↓	0	300	0	220	ns	
T_{DF}	DB → Float Delay from $\overline{RD}$ ↑	20	150	20	120	ns	
T_{RR}	$\overline{RD}$ Width	250		250		ns	

A.C. CHARACTERISTICS—PERIPHERAL (SLAVE) MODE (Continued)

WRITE CYCLE

Symbol	Parameter	8257		8257-5		Unit	Test Conditions
		Min	Max	Min	Max		
T_{AW}	Adr Setup to $\overline{WR}$ ↓	20		20		ns	
T_{WA}	Adr Hold from $\overline{WR}$ ↑	0		0		ns	
T_{DW}	Data Setup to $\overline{WR}$ ↑	200		200		ns	
T_{WD}	Data Hold from $\overline{WR}$ ↑	10		10		ns	
T_{WW}	$\overline{WR}$ Width	200		200		ns	

OTHER TIMING

Symbol	Parameter	8257		8257-5		Unit	Test Conditions
		Min	Max	Min	Max		
T_{RSTW}	Reset Pulse Width	300		300		ns	
T_{RSTD}	Power Supply ↑ (V_{CC}) Setup to Reset ↓	500		500		μs	
T_r	Signal Rise Time		20		20	ns	
T_f	Signal Fall Time		20		20	ns	
T_{RSTS}	Reset to First $\overline{I/O}\overline{WR}$	2		2		t_{CY}	

A.C. CHARACTERISTICS—DMA (MASTER) MODE

8257: T_A = 0°C to 70°C, V_{CC} = 5.0V ±5%, GND = 0V)
8257-5: T_A = 0°C to 70°C, V_{CC} = 5.0V ±10%, GND = 0V)

TIMING REQUIREMENTS

Symbol	Parameter	8257		8257-5		Unit
		Min	Max	Min	Max	
T_{CY}	Cycle Time (Period)	0.320	4	0.320	4	μs
T_θ	Clock Active (High)	120	0.8 T_{CY}	80	0.8 T_{CY}	ns
T_{QS}	DRQ ↑ Setup to CLK ↓ (SI, S4)	120		120		ns
T_{QH}	DRQ ↓ Hold from HLDA ↑ (1)	0		0		ns
T_{HS}	HLDA ↑ or ↓ Setup to CLK ↓ (SI, S4)(7)	100	280	100	280	ns
T_{RS}	READY Setup Time to CLK ↑ (S3, SW)	30		30		ns
T_{RH}	READY Hold Time from CLK ↑ (S3, SW)	30		30		ns

A.C. CHARACTERISTICS—DMA (MASTER) MODE

8257: T_A = 0°C to 70°C, V_{CC} = 5.0V ±5%, GND = 0V

8257-5: T_A = 0°C to 70°C, V_{CC} = 5.0V ±10%, GND = 0V

TIMING RESPONSES

Symbol	Parameter	8257		8257-5		Unit
		Min	Max	Min	Max	
T_{DQ}	HRQ ↑ or ↓ Delay from CLK ↑ (SI, S4) (measured at 2.0V)		160		160	ns
T_{DQ1}	HRQ ↑ or ↓ Delay from CLK ↑ (SI, S4) (measured at 3.3V)(3)		250		250	ns
T_{AEL}	AEN ↑ Delay from CLK ↓ (S1)		300		300	ns
T_{AET}	AEN ↓ Delay from CLK ↑ (SI)		200		200	ns
T_{AEA}	Adr (AB) (Active) Delay from AEN ↑ (S1)(1)	20		20		ns
T_{FAAB}	Adr (AB) (Active) Delay from CLK ↑ (S1)(2)		250		250	ns
T_{AFAB}	Adr (AB) (Float) Delay from CLK ↑ (SI)(2)		150		150	ns
T_{ASM}	Adr (AB) (Stable) Delay from CLK ↑ (S1)(2)		250		250	ns
T_{AH}	Adr (AB) (Stable) Hold from CLK ↑ (S1)(2)	$T_{ASM}-50$		$T_{ASM}-50$		ns
T_{AHR}	Adr (AB) (Valid) Hold from $\overline{RD}$ ↑ (S1, SI)(1)	60		60		ns
T_{AHW}	Adr (AB) (Valid) Hold from $\overline{WR}$ ↑ (S1, SI)(1)	300		300		ns
T_{FADB}	Adr (DB) (Active) Delay from CLK ↑ (S1)(2)		300		300	ns
T_{AFDB}	Adr (DB) (Float) Delay from CLK ↑ (S2)(2)	$T_{STT}+20$	250	$T_{STT}+20$	170	ns
T_{ASS}	Adr (DB) Setup to Adr Stb ↓ (S1–S2)(1)	100		100		ns
T_{AHS}	Adr (DB) (Valid) Hold from Adr Stb ↓ (S2)(1)	20		20		ns
T_{STL}	ADSTB ↑ Delay from CLK ↑ (S1)		200		200	ns
T_{STT}	ADSTB ↓ Delay from CLK ↑ (S2)		140		140	ns
T_{SW}	ADSTB Width (S1–S2)(1)	$T_{CY}-100$		$T_{CY}-100$		ns
T_{ASC}	$\overline{RD}$ ↓ or $\overline{WR}$(Ext) ↓ Delay from Adr Stb ↓ (S2)(1)	25		25		ns
T_{DBC}	$\overline{RD}$ ↓ or $\overline{WR}$(Ext) ↓ Delay from Adr (DB) (Float) (S2)(1)	−10		−10		ns
T_{AK}	DACK ↑ or ↓ Delay from CLK ↓ (S2, S1) and TC/Mark ↑ Delay from CLK ↑ (S3) and TC/Mark ↓ Delay from CLK ↑ (S4)(4)		250		250	ns
T_{DCL}	$\overline{RD}$ ↓ or $\overline{WR}$(Ext) ↓ Delay from CLK ↑ (S2) and WR ↓ Delay from CLK ↑ (S3)(2, 5)		200		200	ns
T_{DCT}	$\overline{RD}$ ↑ Delay from CLK ↓ (S1, SI) and WR ↑ Delay from CLK ↑ (S4)(2, 6)		200		200	ns
T_{FAC}	$\overline{RD}$ or WR (Active) from CLK ↑ (S1)(2)		300		300	ns
T_{AFC}	$\overline{RD}$ or $\overline{WR}$ (Float) from CLK ↑ (S1)(2)		150		150	ns
T_{RWM}	$\overline{RD}$ Width (S2-S1 or SI)(1)	$2T_{CY}+T_\theta-50$		$2T_{CY}+T_\theta-50$		ns
T_{WWM}	$\overline{WR}$ Width (S3-S4)(1)	$T_{CY}-50$		$T_{CY}-50$		ns
T_{WWME}	$\overline{WR}$(Ext) Width (S2-S4)(1)	$2T_{CY}-50$		$2T_{CY}-50$		ns

NOTES:

1. Tracking Parameter.
2. Load = +50 pF.
3. Load = V_{OH} = 3.3V.
4. $\Delta T_{AK} < 50$ ns.
5. $\Delta T_{DCL} < 50$ ns.
6. $\Delta T_{DCT} < 50$ ns.
7. HLDA must remain stable during t_{HS}.

8755A
16,384-BIT EPROM WITH I/O

- **2048 Words x 8 Bits**
- **Single +5V Power Supply (V_{CC})**
- **Directly Compatible with 8085AH**
- **U.V. Erasable and Electrically Reprogrammable**
- **Internal Address Latch**
- **2 General Purpose 8-Bit I/O Ports**
- **Each I/O Port Line Individually Programmable as Input or Output**
- **Multiplexed Address and Data Bus**
- **40-Pin DIP**
- **Available in EXPRESS**
  - **— Standard Temperature Range**
  - **— Extended Temperature Range**

The Intel 8755A is an erasable and electrically reprogrammable ROM (EPROM) and I/O chip to be used in the 8085AH microprocessor systems. The EPROM portion is organized as 2048 words by 8 bits. It has a maximum access time of 450 ns to permit use with no wait states in an 8085AH CPU.

The I/O portion consists of 2 general purpose I/O ports. Each I/O port has 8 port lines, and each I/O port line is individually programmable as input or output.

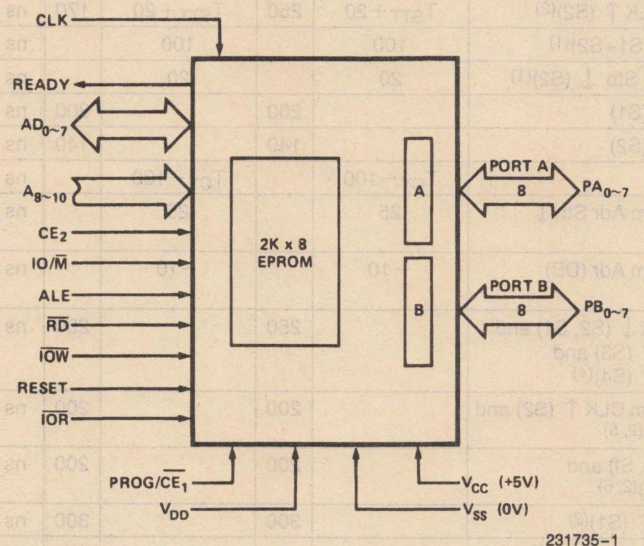


Figure 1. Block Diagram

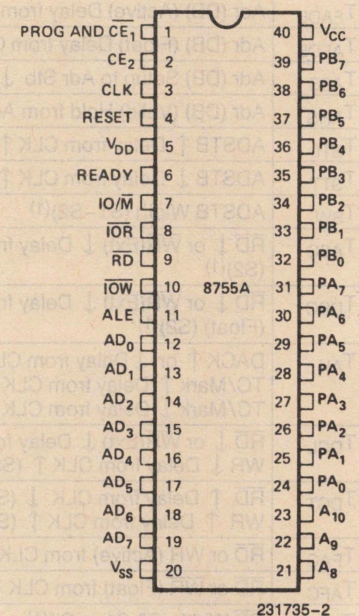


Figure 2. Pin Configuration

November 1986
Order Number: 231735-002

Table 1. Pin Description

Symbol	Type	Name and Function
ALE	I	**ADDRESS LATCH ENABLE:** When Address Latch Enable goes *high,* AD_{0-7}, $IO/\overline{M}$, A_{8-10}, CE_2, and $\overline{CE_1}$ enter the address latches. The signals, (AD, $IO/\overline{M}$, AD_{8-10}, CE_2, $\overline{CE_1}$) are latched in at the trailing edge of ALE.
AD_{0-7}	I	**BIDIRECTIONAL ADDRESS/DATA BUS:** The lower 8 bits of the PROM or I/O address are applied to the bus lines when ALE is high. During an I/O cycle, Port A or B is selected based on the latched value of AD_0. IF $\overline{RD}$ or $\overline{IOR}$ is low when the latched Chip Enables are active, the output buffers present data on the bus.
AD_{8-10}	I	**ADDRESS BUS:** These are the high order bits of the PROM address. They do not affect I/O operations.
PROG/$\overline{CE_1}$ CE_2	I	**CHIP ENABLE INPUTS:** $\overline{CE_1}$ is active low and CE_2 is active high. The 8755A can be accessed only when *both* Chip Enables are active at the time the ALE signal latches them up. If either Chip Enable input is not active, the AD_{0-7}, and READY ouputs will be in a high impedance state. $\overline{CE_1}$ is also used as a programming pin. (See section on programming.)
$IO/\overline{M}$	I	**I/O MEMORY:** If the latched $IO/\overline{M}$ is high when $\overline{RD}$ is low, the output data comes from an I/O port. If it is low the output data comes from the PROM.
$\overline{RD}$	I	**READ:** If the latched Chip Enables are active when $\overline{RD}$ goes low, the AD_{0-7} output buffers are enabled and output either the selected PROM location or I/O port. When both $\overline{RD}$ and $\overline{IOR}$ are high, the AD_{0-7} output buffers are 3-stated.
$\overline{IOW}$	I	**I/O WRITE:** If the latched Chip Enables are active, a low on $\overline{IOW}$ causes the output port pointed to by the latched value of AD_0 to be written with the data on AD_{0-7}. The state of $IO/\overline{M}$ is ignored.
CLK	I	**CLOCK:** The CLK is used to force the READY into its high impedance state after it has been forced low by $\overline{CE_1}$ low, CE_2 high, and ALE high.
READY	O	**READY** is a 3-state output controlled by $\overline{CE_1}$, CE_2, ALE and CLK. READY is forced low when the Chip Enables are active during the time ALE is high, and remains low until the rising edge of the next CLK. (See Figure 6c.)
PA_{0-7}	I/O	**PORT A:** These are general purpose I/O pins. Their input/output direction is determined by the contents of Data Direction Register (DDR). Port A is selected for write operations when the Chip Enables are active and $\overline{IOW}$ is low and a 0 was previously latched from AD_0, AD_1. Read Operation is selected by either $\overline{IOR}$ low and active Chip Enables and AD_0 and AD_1 low, *or* $IO/\overline{M}$ high, $\overline{RD}$ low, active Chip Enables, and AD_0 and AD_1 low.
PB_{0-7}	I/O	**PORT B:** The general purpose I/O port is identical to Port A except that it is selected by a 1 latched from AD_0 and a 0 from AD_1.
RESET	I	**RESET:** In normal operation, an input high on RESET causes all pins in Ports A and B to assume input mode (clear DDR register).
$\overline{IOR}$	I	**I/O READ:** When the Chip Enables are active, a low on $\overline{IOR}$ will output the selected I/O port onto the AD bus. $\overline{IOR}$ low performs the same function as the combination of $IO/\overline{M}$ high and $\overline{RD}$ low. When $\overline{IOR}$ is not used in a system, $\overline{IOR}$ should be tied to V_{CC} ("1").
V_{CC}		**POWER:** +5V supply.
V_{SS}		**GROUND:** Reference.
V_{DD}		**POWER SUPPLY:** V_{DD} is a programming voltage, and must be tied to V_{CC} when the 8755A is being read. For programming, a high voltage is supplied with V_{DD} = 25V, typical. (See section on programming.)

FUNCTIONAL DESCRIPTION

PROM Section

The 8755A contains an 8-bit address latch which allows it to interface directly to MCS®-48 and MCS®-85 processors without additional hardware.

The PROM section of the chip is addressed by the 11-bit address and the Chip Enables. The address, $\overline{CE_1}$ and CE_2 are latched into the address latches on the falling edge of ALE. If the latched Chip Enables are active and IO/$\overline{M}$ is low when $\overline{RD}$ goes low, the contents of the PROM location addressed by the latched address are put out on the AD_{0-7} lines (provided that V_{DD} is tied to V_{CC}).

I/O Section

The I/O section of the chip is addressed by the latched value of AD_{0-1}. Two 8-bit Data Direction Registers (DDR) in 8755A determine the input/output status of each pin in the corresponding ports. A "0" in a particular bit position of a DDR signifies that the corresponding I/O port bit is in the input mode. A "1" in a particular bit position signifies that the corresponding I/O port bit is in the output mode. In this manner the I/O ports of the 8755A are bit-by-bit programmable as inputs or outputs. The table summarizes port and DDR designation. DDR's cannot be read.

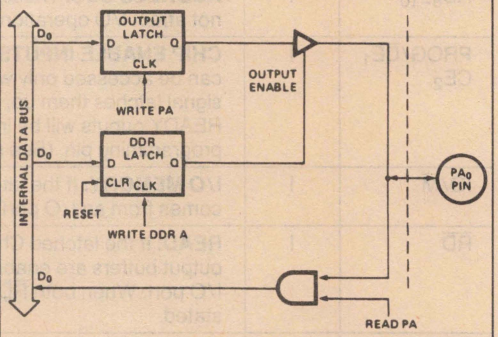
AD_1	AD_0	Selection
0	0	Port A
0	1	Port B
1	0	Port A Data Direction Register (DDR A)
1	1	Port B Data Direction Register (DDR B)

When $\overline{IOW}$ goes low and the Chip Enables are active, the data on the AD_{0-7} is written into I/O port selected by the latched value of AD_{0-1}. During this operation all I/O bits of the selected port are affected, regardless of their I/O mode and the state of IO/$\overline{M}$. The actual output level does not change until $\overline{IOW}$ returns high. (Glitch free output.)

A port can be read out when the latched Chip Enables are active and either $\overline{RD}$ goes low with IO/$\overline{M}$ high, or $\overline{IOR}$ goes low. Both input and output mode bits of a selected port will appear on lines AD_{0-7}.

To clarify the function of the I/O Ports and Data Direction Registers, the following diagram shows the configuration of one bit of PORT A and DDR A. The same logic applies to PORT B and DDR B.

8755A ONE BIT OF PORT A AND DDR A



231735-3

WRITE PA = ($\overline{IOW}$ = 0) • (CHIP ENABLES ACTIVE) • (PORT A ADDRESS SELECTED)
WRITE DDR A = ($\overline{IOW}$ = 0) • (CHIP ENABLES ACTIVE) • (DDR A ADDRESS SELECTED)
READ PA = {(IO/$\overline{M}$ = 1) • ($\overline{RD}$ = 0) + ($\overline{IOR}$ = 0)} • (CHIP ENABLES ACTIVE) • (PORT A ADDRESS SELECTED)

NOTE:
Write PA is not qualified by IO/$\overline{M}$.

Note that hardware RESET or writing a zero to the DDR latch will cause the output latch's output buffer to be disabled, preventing the data in the Output Latch from being passed through to the pin. This is equivalent to putting the port in the input mode. Note also that the data can be written to the Output Latch even though the Output Buffer has been disabled. This enables a port to be initialized with a value prior to enabling the output.

The diagram also shows that the contents of PORT A and PORT B can be read even when the ports are configured as outputs.

ERASURE CHARACTERISTICS

The erasure characteristics of the 8755A are such that erasure begins to occur when exposed to light with wavelengths shorter than approximately 4000 Angstroms (Å). It should be noted that sunlight and certain types of fluorescent lamps have wavelengths in the 3000–4000Å range. Data show that constant exposure to room level fluorescent lighting could erase the typical 8755A in approximately 3 years while it would take approximately 1 week to cause erasure when exposed to direct sunlight. If the 8755A is to be exposed to these types of lighting conditions for extended periods of time, opaque labels are available from Intel which should be placed over the 8755A window to prevent unintentional erasure.

The recommended erasure procedure for the 8755A is exposure to shortwave ultraviolet light which has a wavelength of 2537 Angstroms (Å). The integrated dose (i.e., UV intensity x exposure time) for erasure should be a minimum of 15W-sec/cm^2. The erasure time with this dosage is approximately 15 to 20 minutes using an ultraviolet lamp with a 12000 μW/cm^2 power rating. The 8755A should be placed within one inch from the lamp tubes during erasure. Some lamps have a filter on their tubes and this filter should be removed before erasure.

PROGRAMMING

Initially, and after each erasure, all bits of the EPROM portions of the 8755A are in the "1" state. Information is introduced by selectively programming "0" into the desired bit locations. A programmed "0" can only be changed to a "1" by UV erasure.

The 8755A can be programmed on the Intel Universal Programmer (iUP), and iUPF8744A programming module.

The program mode itself consists of programming a single address at a time, giving a single 50 msec pulse for every address. Generally, it is desirable to have a verify cycle after a program cycle for the same address as shown in the attached timing diagram. **In the verify cycle (i.e., normal memory read cycle) 'V_{DD}' should be at +5V.**

SYSTEM APPLICATIONS

System Interface with 8085AH

A system using the 8755A can use either one of the two I/O Interface techniques:

- Standard I/O
- Memory Mapped I/O

If a standard I/O technique is used, the system can use the feature of both CE_2 and $\overline{CE_1}$. By using a combination of unused address lines A_{11-15} and the Chip Enable inputs, the 8085AH system can use up to 5 8755A's without requiring a CE decoder. See Figure 4.

If a memory mapped I/O approach is used the 8755A will be selected by the combination of both the Chip Enables and IO/$\overline{M}$ using AD_{8-15} address lines. See Figure 3.

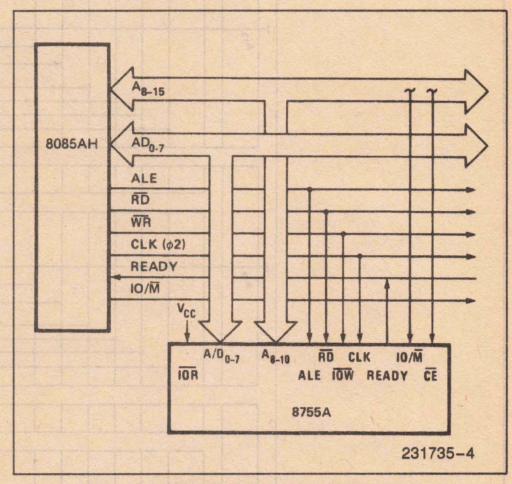
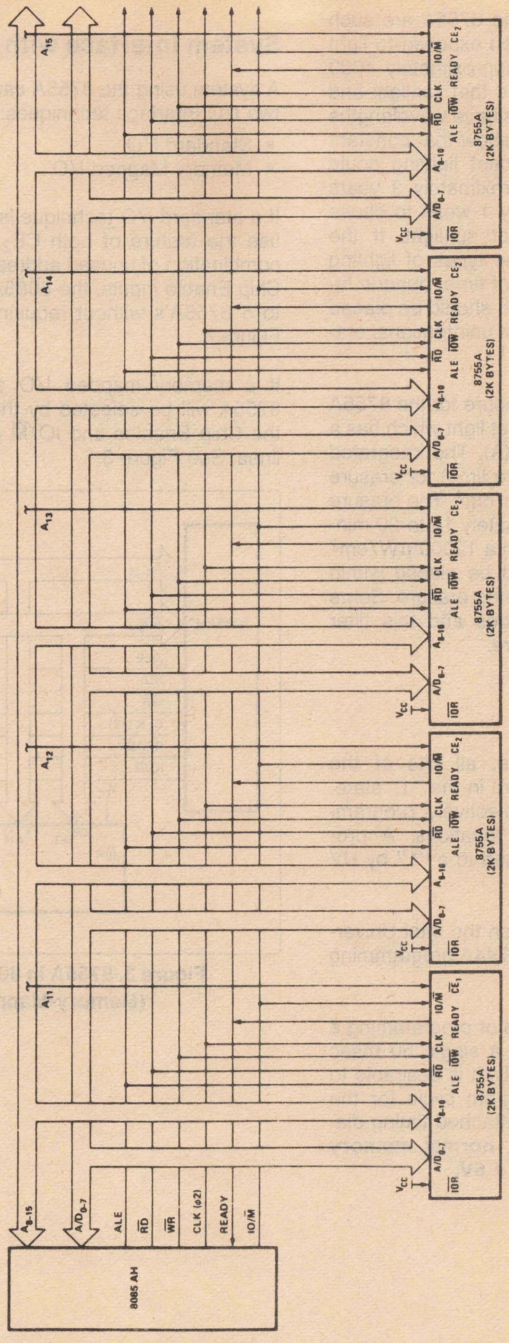


Figure 3. 8755A in 8085AH System (Memory-Mapped I/O)



NOTE:
Use $\overline{CE}_1$ for the first 8755A in the system, and CE_2 for the other 8755A's. Permits up to 5-8755A's in a system without CE decoder.

Figure 4. 8755A in 8085AH System (Standard I/O)

ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias .....0°C to +70°C
Storage Temperature .....−65°C to +150°C
Voltage on any Pin with Respect to Ground.....−0.5V to +7V
Power Dissipation.....1.5W

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS

T_A = 0°C to 70°C, V_{CC} = V_{DD} = 5V ±5%

Symbol	Parameter	Min	Max	Unit	Test Conditions
V_{IL}	Input Low Voltage	−0.5	0.8	V	V_{CC} = 5.0V
V_{IH}	Input High Voltage	2.0	V_{CC} + 0.5	V	V_{CC} = 5.0V
V_{OL}	Output Low Voltage		0.45	V	I_{OL} = 2 mA
V_{OH}	Output High Voltage	2.4		V	I_{OH} = −400 μA
I_{IL}	Input Leakage		10	μA	$V_{SS} \le V_{IN} \le V_{CC}$
I_{LO}	Output Leakage Current		±10	μA	$0.45V \le V_{OUT} \le V_{CC}$
I_{CC}	V_{CC} Supply Current		180	mA	
I_{DD}	V_{DD} Supply Current		30	mA	V_{DD} = V_{CC}
C_{IN}	Capacitance of Input Buffer		10	pF	f_C = 1 μHz
$C_{I/O}$	Capacitance of I/O Buffer		15	pF	f_C = 1 μHz

D.C. CHARACTERISTICS—PROGRAMMING

T_A = 0°C to 70°C, V_{CC} = 5V ±5%, V_{SS} = 0V, V_{DD} = 25V ±1V

Symbol	Parameter	Min	Typ	Max	Unit
V_{DD}	Programming Voltage (during Write to EPROM)	24	25	26	V
I_{DD}	Prog Supply Current		15	30	mA

A.C. CHARACTERISTICS

T_A = 0°C to 70°C, V_{CC} = 5V ±5%

Symbol	Parameter	8755A		Unit
		Min	Max	
t_{CYC}	Clock Cycle Time	320		ns
T_1	CLK Pulse Width	80		ns
T_2	CLK Pulse Width	120		ns
t_f, t_r	CLK Rise and Fall Time		30	ns
t_{AL}	Address to Latch Set Up Time	50		ns
t_{LA}	Address Hold Time after Latch	80		ns
t_{LC}	Latch to READ/WRITE Control	100		ns
t_{RD}	Valid Data Out Delay from READ Control*		170	ns
t_{AD}	Address Stable to Data Out Valid**		450	ns
t_{LL}	Latch Enable Width	100		ns
t_{RDF}	Data Bus Float after READ	0	100	ns
t_{CL}	READ/WRITE Control to Latch Enable	20		ns
t_{CC}	READ/WRITE Control Width	250		ns
t_{DW}	Data in Write Set Up Time	150		ns
t_{WD}	Data in Hold Time after WRITE	30		ns
t_{WP}	WRITE to Port Output		400	ns
t_{PR}	Port Input Set Up Time	50		ns
t_{RP}	Port Input Hold Time to Control	50		ns
t_{RYH}	READY HOLD Time to Control	0	160	ns
t_{ARY}	ADDRESS (CE) to READY		160	ns
t_{RV}	Recovery Time between Controls	300		ns
t_{RDE}	READ Control to Data Bus Enable	10		ns

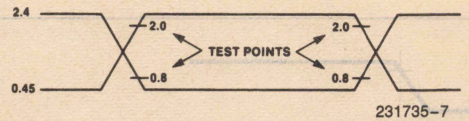
NOTES:
C_{LOAD} = 150 pF.
*Or $T_{AD} - (T_{AL} + T_{LC})$, whichever is greater.
**Defines ALE to Data Out Valid in conjunction with T_{AL}.

A.C. CHARACTERISTICS—PROGRAMMING

T_A = 0°C to 70°C, V_{CC} = 5V ±5%, V_{SS} = 0V, V_{DD} = 25V ±1V

Symbol	Parameter	Min	Typ	Max	Unit
t_{PS}	Data Setup Time	10			ns
t_{PD}	Data Hold Time	0			ns
t_S	Prog Pulse Setup Time	2			μs
t_H	Prog Pulse Hold Time	2			μs
t_{PR}	Prog Pulse Rise Time	0.01	2		μs
t_{PF}	Prog Pulse Fall Time	0.01	2		μs
t_{PRG}	Prog Pulse Width	45	50		ms

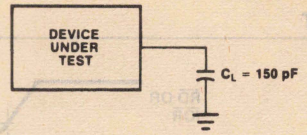
A.C. TESTING INPUT, OUTPUT WAVEFORM



231735-7

A.C. Testing: Inputs are driven at 2.4V for a Logic "1" and 0.45V for a Logic "0". Timing Measurements are made at 2.0V for a Logic "1" and 0.8V for a Logic "0".

A.C. TESTING LOAD CIRCUIT

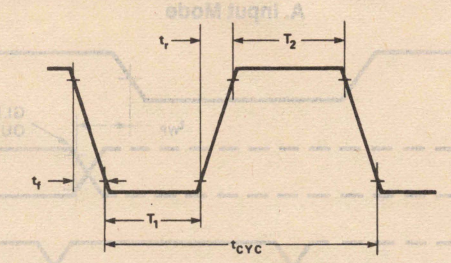


231735-8

C_L = 150 pF
C_L Includes Jig Capacitance

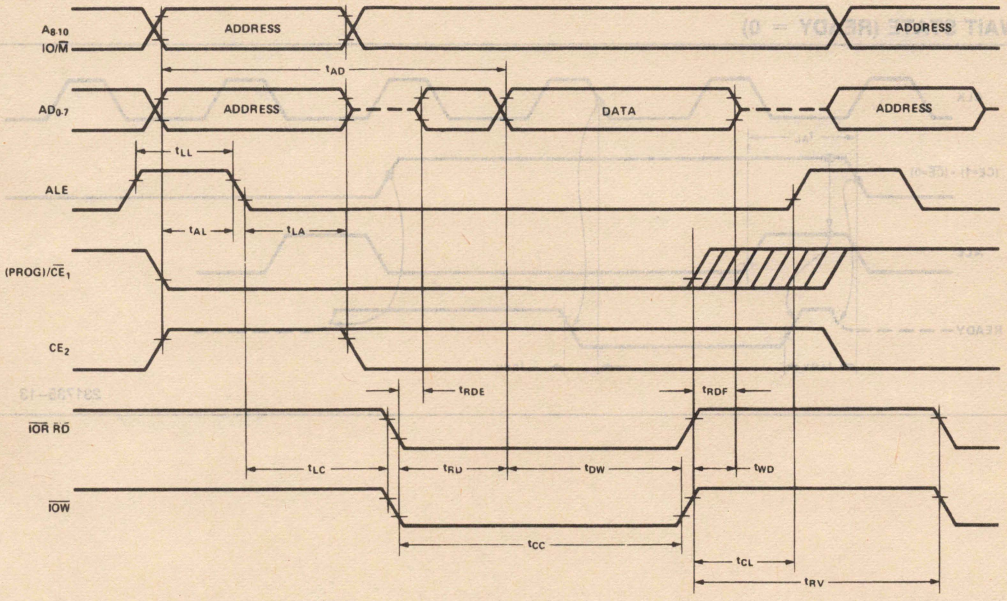
WAVEFORMS

CLOCK SPECIFICATION FOR 8755A



231735-9

PROM READ, I/O READ AND WRITE

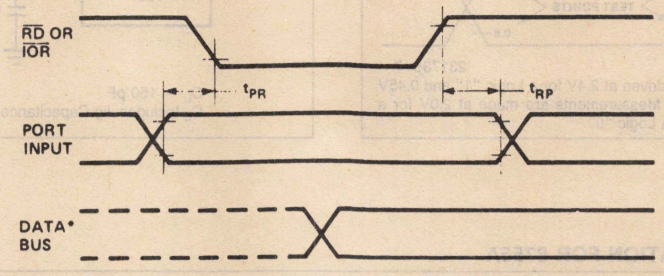


231735-10

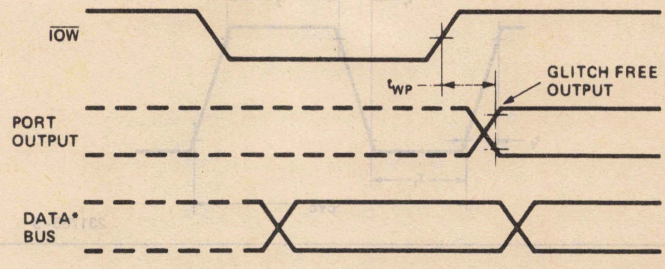
Please note that $\overline{CE_1}$ **must remain low for the entire cycle.**

WAVEFORMS (Continued)

I/O PORT

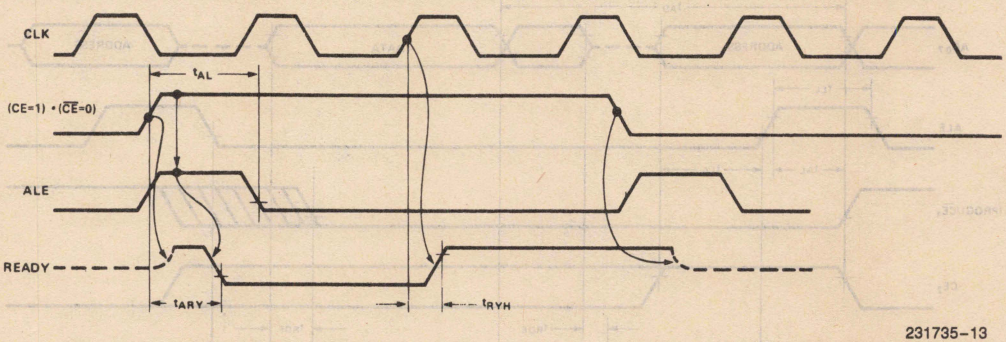


A. Input Mode



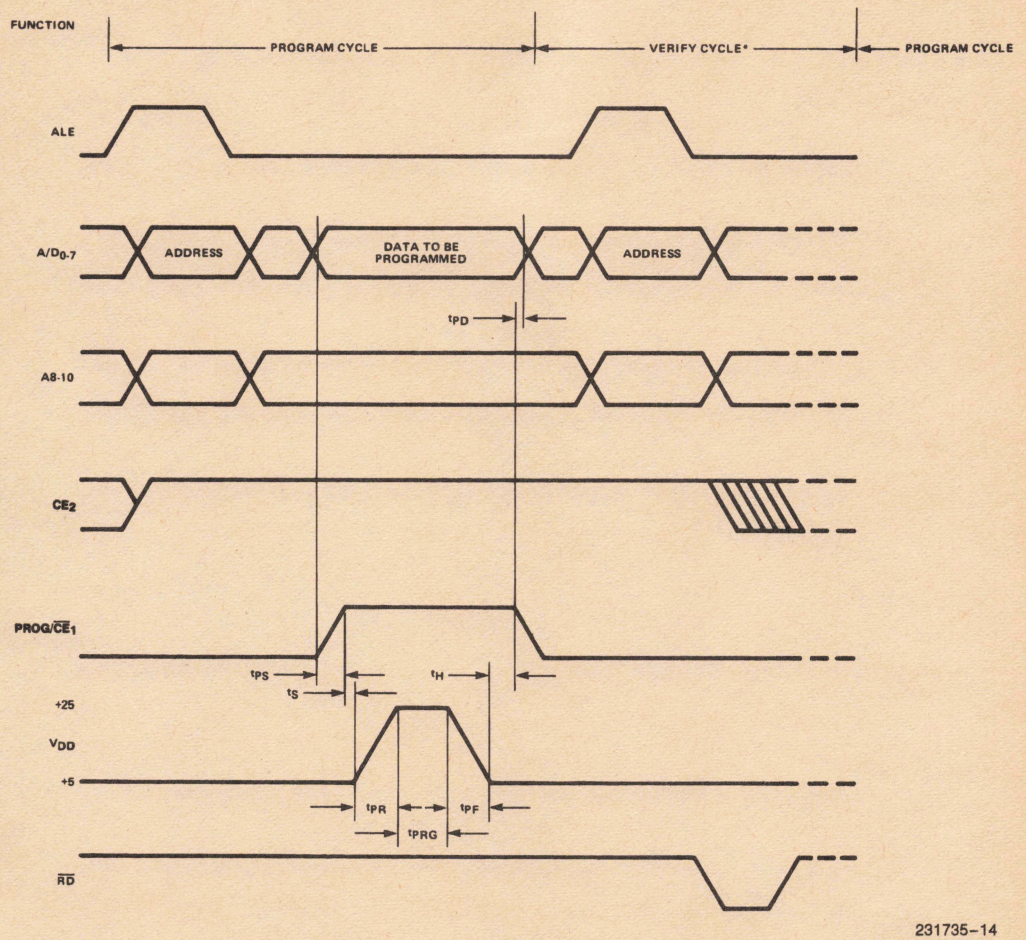
B. Output Mode

WAIT STATE (READY = 0)

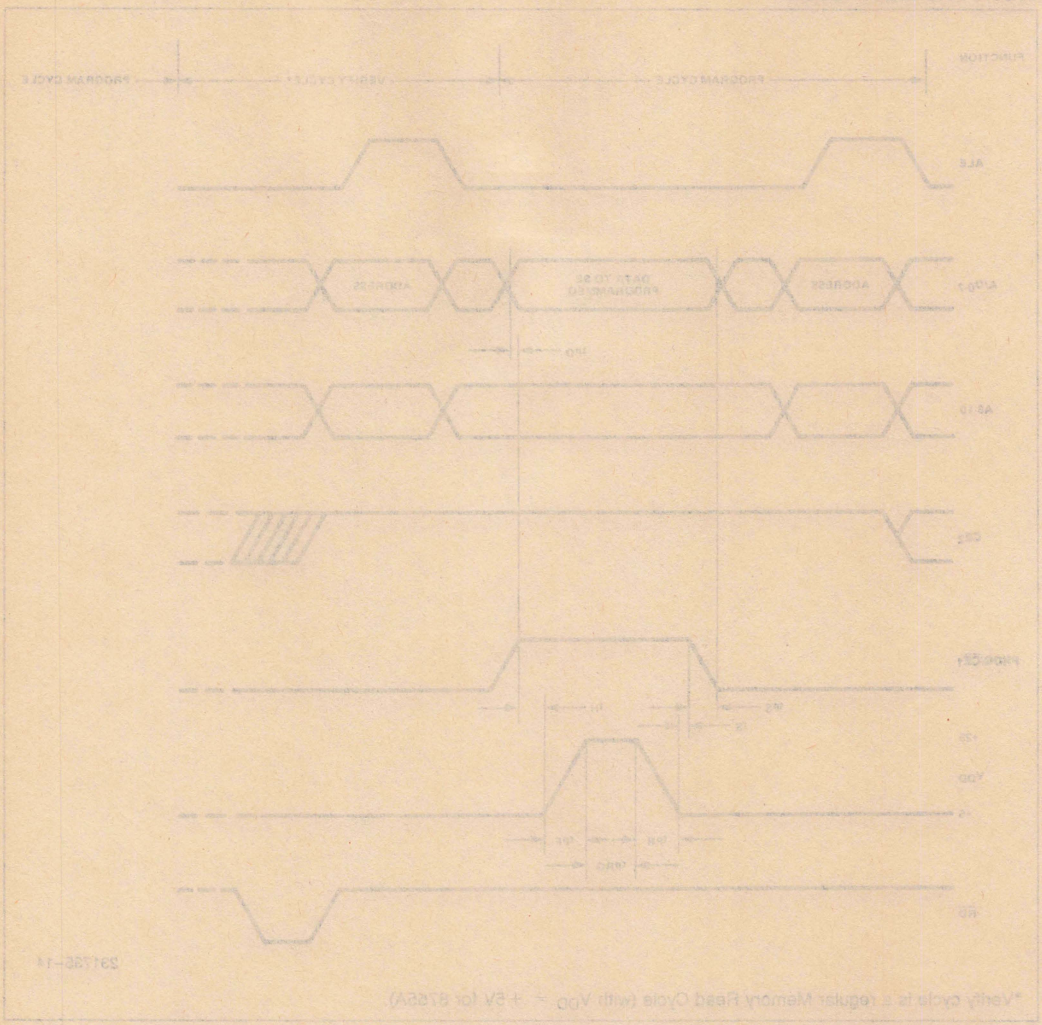


WAVEFORMS (Continued)

8755A PROGRAM MODE



*Verify cycle is a regular Memory Read Cycle (with $V_{DD} = +5V$ for 8755A).



MCS®-96 Architectural Overview

12

MCS®-96 ARCHITECTURAL OVERVIEW

There are two groups of parts within the MCS®-96 family: the standard 8X9X parts and the 8X9XBH parts. There are several enhancements on the 8X9XBH parts that are not on the 8X9X parts. This manual is written about the 8X9XBH parts, generically referred to as an 8096BH. Where the standard 8X9X parts differ from the 8096BH, notations will be made.

The 8096BH can be separated into several sections for the purpose of describing its operation. There is a 16-bit CPU, a programmable High Speed I/O Unit, an analog to digital converter, a serial port, and a Pulse Width Modulated (PWM) output for digital to analog conversion. In addition to these functional units, there are some sections which support overall operation of the chip such as the clock generator. The CPU and the programmable I/O make the 8096BH very different from any other microcontroller. Let us first examine the CPU.

1.0 CPU OPERATION

The major components of the CPU on the 8096BH are the Register File and the RALU. Communication with the outside world is done through either the Special Function Registers (SFRs) or the Memory Controller. The RALU (Register/Arithmetic Logic Unit) does not use an accumulator, it operates directly on the 256-byte register space made up of the Register File and the SFRs. Efficient I/O operations are possible by directly controlling the I/O through the SFRs. The main benefits of this structure are the ability to quickly change context, the absence of accumulator bottleneck, and fast throughput and I/O times.

1.1 CPU Buses

A "Control Unit" and two busses connect the Register File and RALU. Figure 1 shows the CPU with its

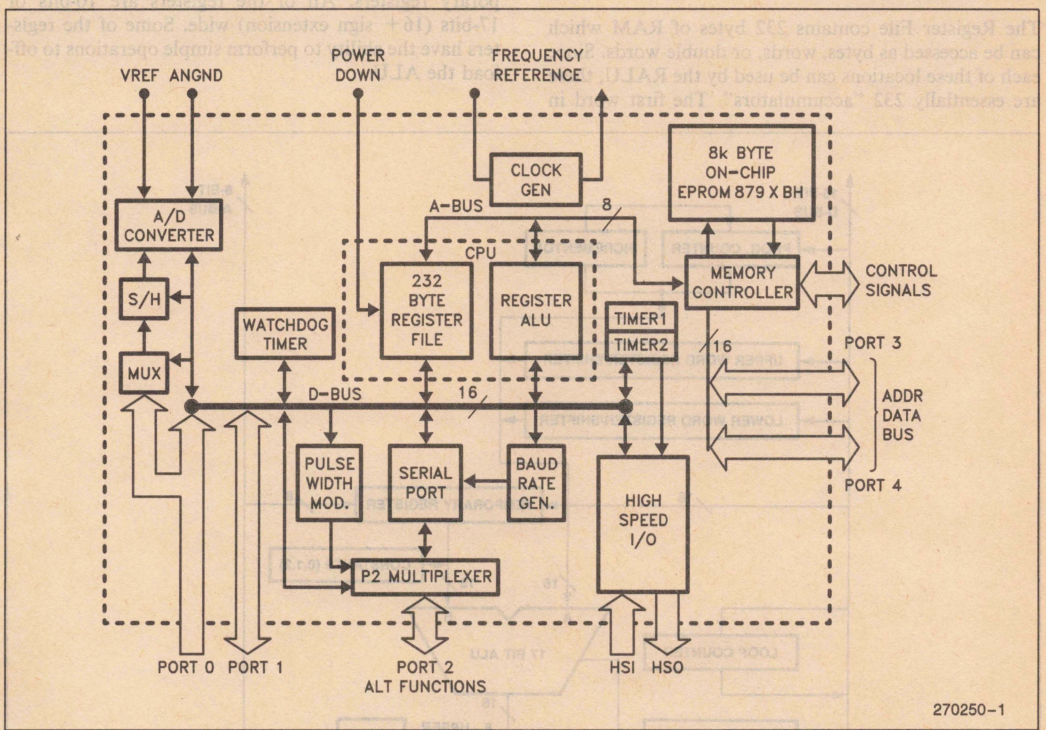


Figure 1. Block Diagram

major bus connections. The two buses are the "A-Bus" which is 8-bits wide, and the "D-Bus" which is 16-bits wide. The D-Bus transfers data only between the RALU and the Register File or Special Function Registers (SFRs). The A-Bus is used as the address bus for the above transfers or as a multiplexed address/data bus connecting to the "Memory Controller". Any accesses of either the internal ROM or external memory are done through the Memory Controller.

Within the memory controller is a slave program counter (Slave PC) which keeps track of the PC in the CPU. By having most program fetches from memory referenced to the slave PC, the processor saves time as addresses seldom have to be sent to the memory controller. If the address jumps sequence then the slave PC is loaded with a new value and processing continues. Data fetches from memory are also done through the memory controller, but the slave PC is bypassed for this operation.

1.2 CPU Register File

The Register File contains 232 bytes of RAM which can be accessed as bytes, words, or double-words. Since each of these locations can be used by the RALU, there are essentially 232 "accumulators". The first word in the Register File is reserved for use as the stack pointer so it can not be used for data when stack manipulations are taking place. Addresses for accessing the Register File and SFRs are temporarily stored in two 8-bit address registers by the CPU hardware.

1.3 RALU Control

Instructions to the RALU are taken from the A-Bus and stored temporarily in the instruction register. The Control Unit decodes the instructions and generates the correct sequence of signals to have the RALU perform the desired function. Figure 1 shows the instruction register and the control unit.

1.4 RALU

Most calculations performed by the 8096BH take place in the RALU. The RALU, shown in Figure 2, contains a 17-bit ALU, the Program Status Word (PSW), the Program Counter (PC), a loop counter, and three temporary registers. All of the registers are 16-bits or 17-bits (16+ sign extension) wide. Some of the registers have the ability to perform simple operations to offload the ALU.

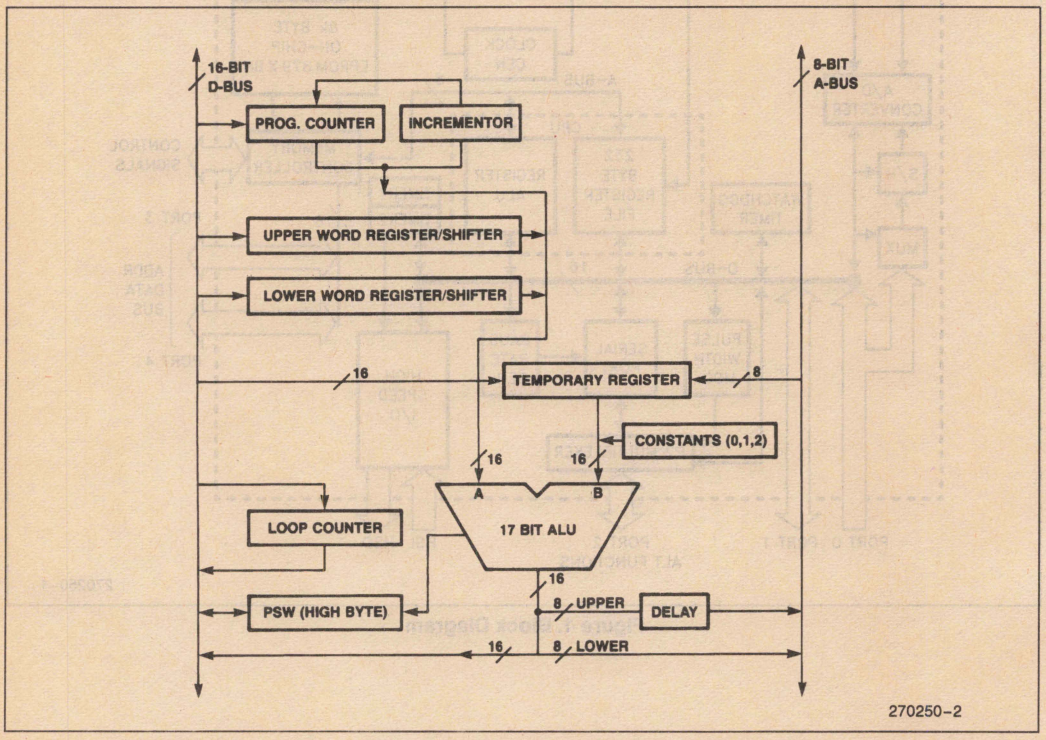


Figure 2. RALU Block Diagram

A separate increment is used for the PC; however, jumps must be handled through the ALU. Two of the temporary registers have their own shift logic. These registers are used for the operations which require logical shifts, including Normalize, Multiply, and Divide. The "Lower Word" register is used only when doubleword quantities are being shifted, the "Upper Word" register is used whenever a shift is performed or as a temporary register for many instructions. Repetitive shifts are counted by the 5-bit "Loop Counter".

A temporary register is used to store the second operand of two operand instructions. This includes the multiplier during multiplications and the divisor during divisions. To perform subtractions, the output of this register can be complemented before being placed into the "B" input of the ALU.

The DELAY shown in Figure 2 is used to convert the 16-bit bus into an 8-bit bus. This is required as all addresses and instructions are carried on the 8-bit A-Bus. Several constants, such as 0, 1 and 2 are stored in the RALU for use in speeding up certain calculations. These come in handy when the RALU needs to make a 2's complement number or perform an increment or decrement instruction.

1.5 Internal Timing

The 8096BH requires an input clock frequency of between 6.0 MHz and 12 MHz to function. This frequency can be applied directly to XTAL1. Alternatively, since XTAL1 and XTAL2 are inputs and outputs of an inverter, it is also possible to use a crystal to generate the clock. A block diagram of the oscillator section is shown in Figure 3. Details of the circuit and suggestions for its use can be found in Section 1 of the Hardware Design chapter.

The crystal or external oscillator frequency is divided by 3 to generate the three internal timing phases as shown in Figure 4. Each of the internal phases repeat every 3 oscillator periods: 3 oscillator periods are referred to as one "state time", the basic time measurement for 8096BH operations. Most internal operations are synchronized to either Phase A, B or C, each of which have a 33% duty cycle. Phase A is represented externally by CLKOUT, a signal available on the 68-pin part. Phases B and C are not available externally. The relationships of XTAL1, CLKOUT, and Phases A, B, and C are shown in Figure 4. It should be noted that propagation delays have not been taken into account in this diagram. Details on these and other timing relationships can be found in the Hardware Design chapter.

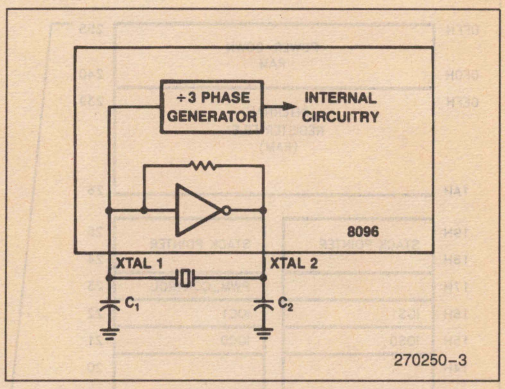


Figure 3. Block Diagram of Oscillator

The $\overline{\text{RESET}}$ line can be used to start the 8096BH at an exact time to provide for synchronization of test equipment and multiple chip systems. Use of this feature is fully explained under RESET, Section 13.

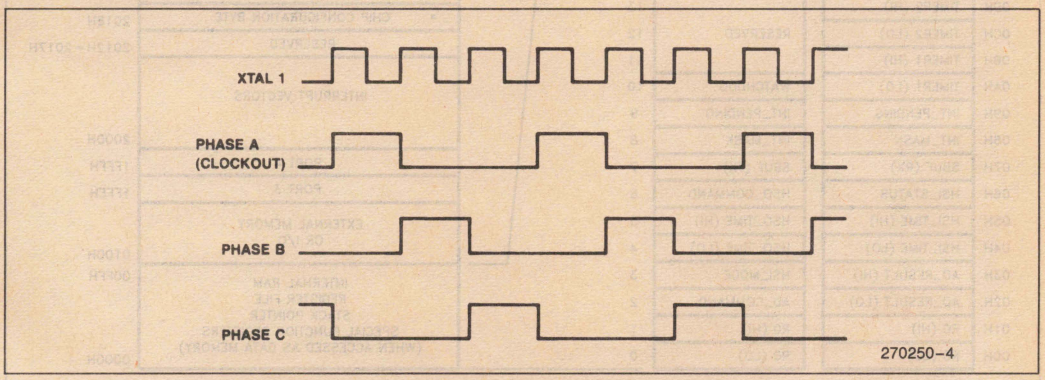


Figure 4. Internal Timings Relative to XTAL 1

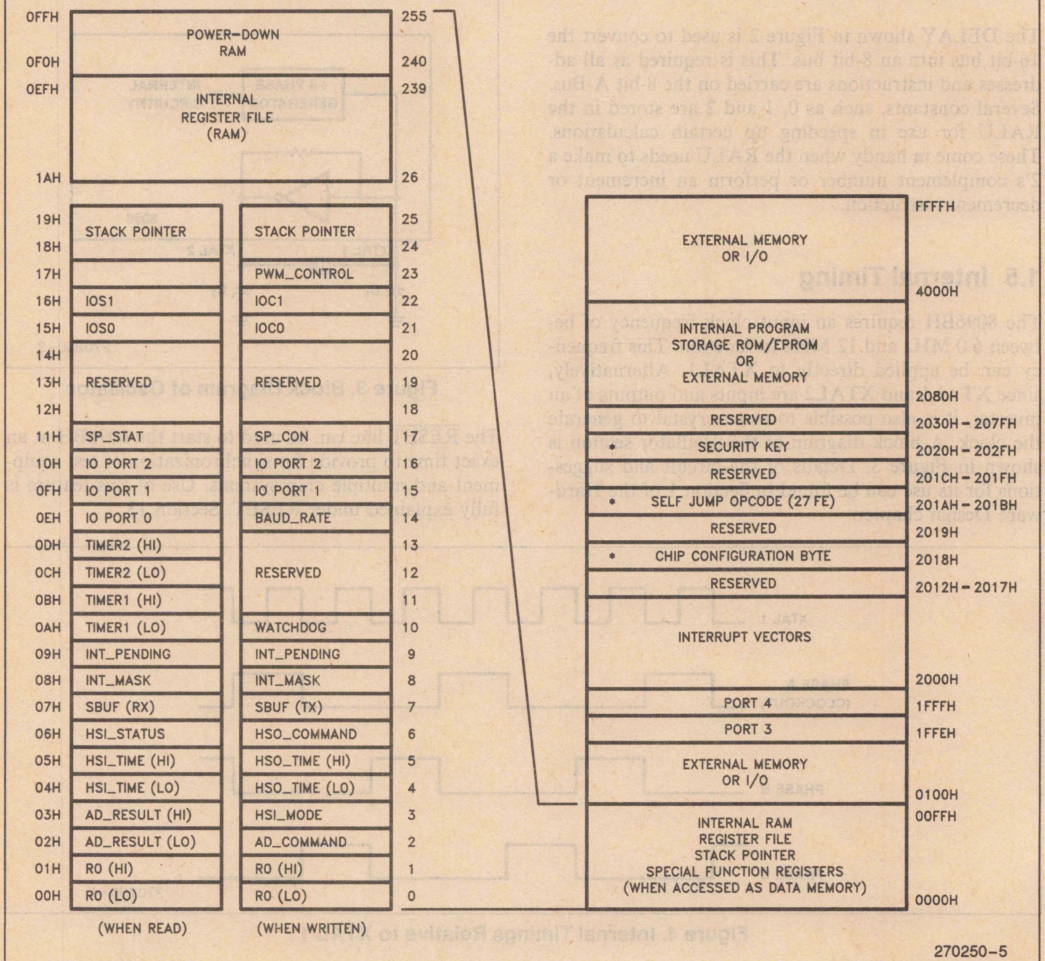
2.0 MEMORY SPACE

The addressable memory space on the 8096BH consists of 64K bytes, most of which is available to the user for program or data memory. Locations which have special purposes are 0000H through 00FFH and 1FFEH through 2080H. All other locations can be used for either program or data storage or for memory mapped peripherals. A memory map is shown in Figure 5.

2.1 Register File

Locations 00H through 0FFH contain the Register File and Special Function Registers, (SFRs). No code can be executed from this internal RAM section. If an attempt to execute instructions from locations 000H through 0FFH is made, the instructions will be fetched from *external* memory. This section of external memory is reserved for use by Intel development tools. Execution of a nonmaskable interrupt (NMI) will force a call to external location 0000H, therefore, the NMI instruction is also reserved for Intel development tools.

The RALU can operate on any of the 256 internal register locations. Locations 00H through 17H are used to access the SFRs. Locations 18H and 19H contain the stack pointer. These are not SFRs, and may be used as standard RAM if stack operations are not being performed. The stack pointer must be initialized by the



NOTE:
*Registers marked by an asterisk are not present on 8X9X devices

Figure 5. Memory Map

user program and can point anywhere in the 64K memory space. The stack builds down. There are no restrictions on the use of the remaining 230 locations except that code cannot be executed from them.

2.2 Special Function Registers

All of the I/O on the 8096BH is controlled through the SFRs. Many of these registers serve two functions; one if they are read from, the other if they are written to. Figure 5 shows the locations and names of these registers. A summary of the capabilities of each of these registers is shown in Figure 6, with complete descriptions reserved for later sections.

There are several restrictions on using special function registers.

Neither the source or destination addresses of the Multiply and Divide instructions can be a writable special function register.

These registers may not be used as base or index registers for indirect or indexed instructions.

These registers can only be accessed as bytes unless otherwise specified in Figure 6. Note that some of these registers can only be accessed as words, and not as bytes.

Within the SFR space are several registers labeled "RESERVED". These registers are reserved for future expansion and test purposes. Operations should not be performed with these registers as reads from them and writes to them may produce unexpected results. For example, in some versions of the 8096 writing to location 0CH will set both timers to 0FFFXH. This may not be the case in future products, so it should not be used as a feature.

2.3 Power Down

The upper 16 RAM locations (0F0H through 0FFH) receive their power from the VPD pin. If it is desired to keep the memory in these locations alive during a power down situation, one need only keep voltage on the VPD pin. The current required to keep the RAM alive is approximately 1 milliamp (refer to the data sheet for the exact specification). Both VCC and VPD must have power applied for normal operation. If VPD is not applied the power down RAM will not function properly, even if VCC is applied.

To place the 8096BH into a power down mode, the $\overline{\text{RESET}}$ pin is pulled low. Two state times later the part will be in reset. This is necessary to prevent the part from writing into RAM as the power goes down. The power may now be removed from the VCC pin, the VPD pin must remain within specifications. The 8096BH can remain in this state for any amount of time and the 16 RAM bytes will retain their values.

To bring the 8096BH out of power down, $\overline{\text{RESET}}$ is held low while VCC is applied. Two state times after the oscillator has stabilized, the $\overline{\text{RESET}}$ pin can be pulled high. *On the 8X9X devices the back-bias generator must also stabilize. This requires approximately 1 millisecond.* The 8096BH will begin to execute code at location 02080H 10 state times after $\overline{\text{RESET}}$ is pulled high. Figure 7 shows a timing diagram of the power down sequence. To ensure that the 2 state time minimum reset time (synchronous with CLKOUT) is met, it is recommended that 10 XTAL1 cycles be used. Suggestions for actual hardware connections are given in the Hardware Design Chapter. Reset is discussed in Section 13.

To determine if a reset is a return from power down or a complete cold start a "key" can be written into power-down RAM while the part is running. This key can be checked on reset to determine which type of reset has occurred. In this way the validity of the power-down RAM can be verified. The length of this key determines the probability that this procedure will work, however, there is always a statistical chance that the RAM will power up with a replica of the key.

Under most circumstances, the power-fail indicator which is used to initiate a power-down condition must come from the unfiltered, unregulated section of the power supply. The power supply must have sufficient storage capacity to operate the 8096BH until it has completed its reset operation.

Register	Description	Section
R0	Zero Register — Always reads as a zero, useful for a base when indexing and as a constant for calculations and compares.	3
AD_RESULT	A/D Result Hi/Low — Low and high order Results of the A/D converter (byte read only)	8
AD_COMMAND	A/D Command Register — Controls the A/D	8
HSI_MODE	HSI Mode Register — Sets the mode of the High Speed Input unit.	6
HSI_TIME	HSI Time Hi/Lo — Contains the time at which the High Speed Input unit was triggered. (word read only)	6
HSO_TIME	HSO Time Hi/Lo — Sets the time or count for the High Speed Output to execute the command in the Command Register. (word write only)	7
HSO_COMMAND	HSO Command Register — Determines what will happen at the time loaded into the HSO Time registers.	7
HSI_STATUS	HSI Status Registers — Indicates which HSI pins were detected at the time in the HSI Time registers and the current state of the pins.	6
SBUF (TX)	Transmit buffer for the serial port, holds contents to be outputted.	9
SBUF (RX)	Receive buffer for the serial port, holds the byte just received by the serial port.	9
INT_MASK	Interrupt Mask Register — Enables or disables the individual interrupts.	4
INT_PENDING	Interrupt Pending Register — Indicates that an interrupt signal has occurred on one of the sources and has not been serviced.	4
WATCHDOG	Watchdog Timer Register — Written to periodically to hold off automatic reset every 64K state times.	12
TIMER1	Timer 1 Hi/Lo — Timer 1 high and low bytes. (word read only)	5
TIMER2	Timer 2 Hi/Lo — Timer 2 high and low bytes. (word read only)	5
IOPORT0	Port 0 Register — Levels on pins of port 0.	10
BAUD_RATE	Register which determines the baud rate, this register is loaded sequentially.	9
IOPORT1	Port 1 Register — Used to read or write to Port 1.	10
IOPORT2	Port 2 Register — Used to read or write to Port 2.	10
SP_STAT	Serial Port Status — Indicates the status of the serial port.	9
SP_CON	Serial Port Control — Used to set the mode of the serial port.	9
IOS0	I/O Status Register 0 — Contains information on the HSO status	11
IOS1	I/O Status Register 1 — Contains information on the status of the timers and of the HSI.	11
IOC0	I/O Control Register 0 — Controls alternate functions of HSI pins, Timer 2 reset sources and Timer 2 clock sources.	11
IOC1	I/O Control Register 1 — Controls alternate functions of Port 2 pins, timer interrupts and HSI interrupts.	11
PWM_CONTROL	Pulse Width Modulation Control Register — Sets the duration of the PWM pulse.	8

Figure 6. SFR Summary

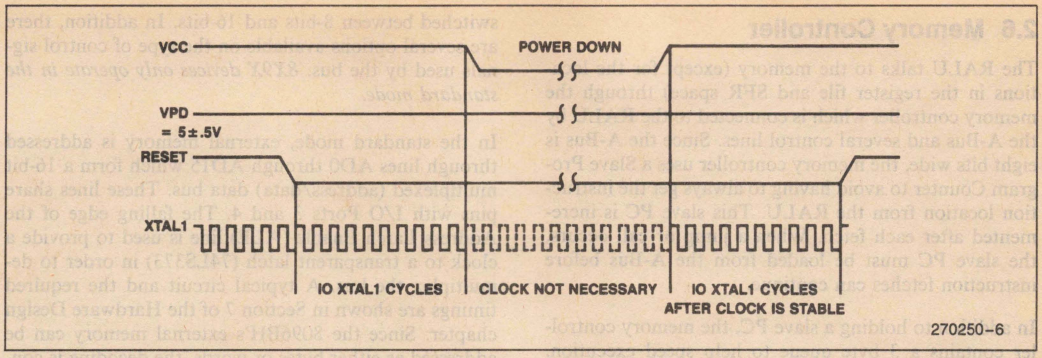


Figure 7. Power Down Timing

2.4 Reserved Memory Spaces

A listing of locations with special significance is shown in Figure 8. The locations marked "Reserved" are reserved by Intel for use in testing or future products. They must be filled with the Hex value FFH to insure compatibility with future parts.

Locations 1FFEH and 1FFFH are reserved for Ports 3 and 4 respectively. This is to allow easy reconstruction of these ports if external memory is used in the system. An example of reconstructing the I/O ports is given in section 7 of the Hardware Design chapter. If ports 3 and 4 are not going to be reconstructed, these locations can be treated as any other external memory location.

The 9 interrupt vectors are stored in locations 2000H through 2011H. The 9th vector is used by Intel development systems, as explained in Section 4.

Locations 2012H through 2017H are reserved for future use. Location 2018H is the Chip Configuration byte which will be discussed in the next section. The Jump-To-Self opcodes at locations 201AH and 201BH are provided for EPROM programming as detailed in the Hardware Design chapter. Locations 2020H through 202FH are the security key used with the ROM Lock feature which will be discussed in the next section. All unspecified addresses in locations 2000H through 207FH, including those marked Reserved, should be considered reserved for use by Intel.

Resetting the 8096BH causes instructions to be fetched starting from location 2080H. This location was chosen to allow a system to have up to 8K of RAM continuous with the register file. Further information on reset can be found in Section 13.

0000H-	0017H	Register Mapped I/O (SFRs)
0018H-	0019H	Stack Pointer
1FFEH-	1FFFH	Ports 3 and 4
2000H-	2011H	Interrupt Vectors
2012H-	2017H	Reserved
2018H		Chip Configuration Byte
2019H		Reserved
201AH-	201BH	"Jump to Self" Opcode (27 FE)
201CH-	201FH	Reserved
2020H-	202FH	Security Key
2030H-	207FH	Reserved
2080H		Reset Location

Figure 8. Registers with Special Significance

2.5 Internal ROM and EPROM

When a ROM part is ordered, or an EPROM part is programmed, the internal memory locations 2080H through 3FFFH are user specified, as are the interrupt vectors, Chip Configuration Register and Security Key in locations 2000H through 202FH.

Instruction and data fetches from the internal ROM or EPROM occur only if the part has a ROM or EPROM, $\overline{EA}$ is tied high, and the address is between 2000H and 3FFFH. At all other times data is accessed from either the internal RAM space or external memory and instructions are fetched from external memory. The $\overline{EA}$ pin is latched on $\overline{RESET}$ rising. Information on programming EPROMs can be found in Section 10 of the Hardware Design chapter.

2.6 Memory Controller

The RALU talks to the memory (except for the locations in the register file and SFR space) through the memory controller which is connected to the RALU by the A-Bus and several control lines. Since the A-Bus is eight bits wide, the memory controller uses a Slave Program Counter to avoid having to always get the instruction location from the RALU. This slave PC is incremented after each fetch. When a jump or call occurs, the slave PC must be loaded from the A-Bus before instruction fetches can continue.

In addition to holding a slave PC, the memory controller contains a 3 byte queue to help speed execution. This queue is transparent to the RALU and to the user unless wait states are forced during external bus cycles. The instruction execution times shown in Section 14.8 show the normal execution times with no wait states added and the 16-bit bus selected. Reloading the slave PC and fetching the first byte of the new instruction stream takes 4 state times. This is reflected in the jump taken/not-taken times shown in the table.

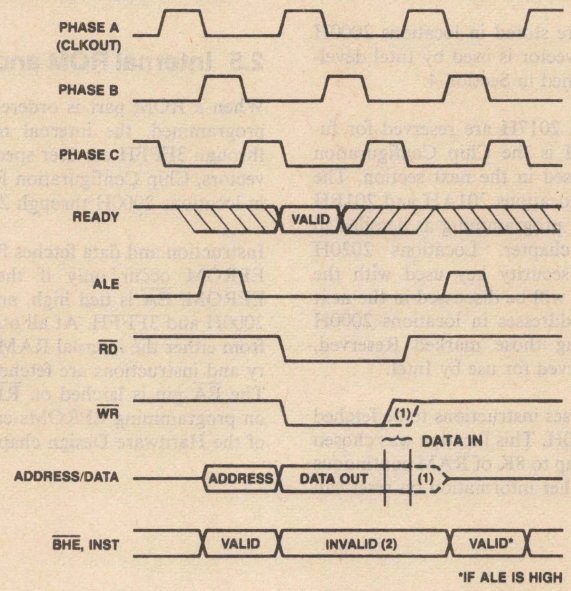
2.7 System Bus

There are several operating modes on the 8096BH. The standard bus mode uses a 16-bit multiplexed address/data bus. Other bus modes include an 8-bit mode and a mode in which the bus size can dynamically be switched between 8-bits and 16-bits. In addition, there are several options available on the type of control signals used by the bus. *8X9X devices only operate in the standard mode.*

In the standard mode, external memory is addressed through lines AD0 through AD15 which form a 16-bit multiplexed (address/data) data bus. These lines share pins with I/O Ports 3 and 4. The falling edge of the Address Latch Enable (ALE) line is used to provide a clock to a transparent latch (74LS373) in order to demultiplex the bus. A typical circuit and the required timings are shown in Section 7 of the Hardware Design chapter. Since the 8096BH's external memory can be addressed as either bytes or words, the decoding is controlled with two lines, Bus High Enable ($\overline{BHE}$) and Address/Data Line 0 (AD0). *On the 8X9X devices the $\overline{BHE}$ line must be transparently latched, just as the addresses are.*

To avoid confusion during the explanation of the memory system it is reasonable to give names to the demultiplexed address/data signals. The address signals will be called MA0 through MA15 (Memory Address), and the data signals will be called MD0 through MD15 (Memory Data).

When $\overline{BHE}$ is active (low), the memory connected to the high byte of the data bus should be selected. When MA0 is low the memory connected to the low byte of



NOTES:
1. These sections are not valid on 8X9X devices (16-bit Mode)
2. $\overline{BHE}$ is valid for the entire bus cycle on the 8096BH

270250-7

Figure 9. External Memory Timings

the data bus should be selected. In this way accesses to a 16-bit wide memory can be to the low (even) byte only (MA0=0, $\overline{BHE}$=1), to the high (odd) byte only (MA0=1, $\overline{BHE}$=0), or to both bytes (MA0=0, $\overline{BHE}$=0). When a memory block is being used only for reads, $\overline{BHE}$ and MA0 need not be decoded.

TIMINGS

Figure 9 shows the idealized waveforms related to the following description of external memory manipulations. For exact timing specifications please refer to the latest data sheet. When an external memory fetch begins, the address latch enable (ALE) line rises, the address is put on AD0–AD15 and $\overline{BHE}$ is set to the required state. ALE then falls, the address is taken off the pins, and the $\overline{RD}$ (Read) signal goes low. When $\overline{RD}$ falls, external memory should present its data to the 8096BH.

READ

The data from the external memory must be on the bus and stable for a minimum of the specified set-up time before the rising edge of $\overline{RD}$. The rising edge of $\overline{RD}$ latches the information into the 8096BH. If the read is for data, the INST pin will be low when the address is valid, if it is for an instruction the INST pin will be high during this time. The 48-lead part does not have the INST pin. The INST pin will be low for the Chip Configuration Byte and Interrupt Vector fetches.

WRITE

Writing to external memory requires timings that are similar to those required when reading from it. The main difference is that the write ($\overline{WR}$) signal is used instead of the $\overline{RD}$ signal. The timings are the same until the falling edge of the $\overline{WR}$ line. At this point the 8096BH removes the address and places the data on the bus. When the $\overline{WR}$ line goes high the data should be latched to the external memory. In systems which can write to byte locations, the AD0 and $\overline{BHE}$ lines must be used to decode $\overline{WR}$ into WRite to Low byte ($\overline{WRL}$) and WRite to High byte ($\overline{WRH}$) signals. INST is always low during a write, as instructions cannot be written. The exact timing specifications for memory accesses can be found in the data sheet.

READY

A ready line is available on the 8096BH to extend the width of the RD and WR pulses in order to allow access of slow memories or for DMA purposes. If the READY line is low by the specified time after ALE falls, the 8096BH will hold the bus lines to their values at the falling edge of CLKOUT. When the READY line rises the bus cycle will continue with the next falling edge of CLKOUT.

Since the bus is synchronized to CLKOUT, it can be held only for an integral number of state times. If more than TYLYH nanoseconds are added the processor will act unpredictably.

There are several set-up and hold times associated with the READY signal. If these timings are not met, the part may not respond with the proper number of wait states.

For falling edges of READY, sampling is done internally on the falling edge of Phase A. Since Phase A generates CLKOUT, (after some propagation delay) the sample will be taken prior to CLKOUT falling. The timing specification for this is given as TLLYV, the time between when ALE falls and READY must be valid. If READY changes between TLLYV max and the falling edge of CLKOUT (TLLYH MIN on 48-lead devices) it would be possible to have the READY signal transitioning as it is being sampled.

This situation could cause a metastable condition which could make the device operate unpredictably.

For the rising edge of READY, sampling is done internally on the rising edge of Phase A. The rising edge logic is fully synchronized, so it is not possible to cause a metastable condition once the device is in a valid not-ready condition. To cause one wait state to occur the rising edge of READY must occur before TLLYH MAX after ALE falls. If the signal is brought up after this time two wait states may occur. If two wait states are desired, READY should be brought high within the TLLYH specification + 3 Tosc. Additional wait states can be caused by adding additional state times to the READY low time. The maximum amount of time that a device may be held not-ready is specified as TYLYH.

The 8096BH has the ability to internally limit the number of wait states to 1, 2, or 3 as determined by the value in the Chip Configuration Register, (CCR). Using the CCR for ready timing is discussed at the end of this section. If a ready limit is set, the TLLYH MAX specification is not used. *The 8X9X devices do not have internal ready control.*

OPERATING MODES

The 8096BH supports a variety of options to simplify memory systems, interfacing requirements and ready control. Bus flexibility is provided by allowing selection of bus control signal definitions and runtime selection of the external bus width. In addition, several ready control modes are available to simplify the external hardware requirements for accessing slow devices. The Chip Configuration Register (CCR) is used to store the operating mode information.

Since there is no CCR on 8X9X devices, they can only be configured in the standard mode. This is the mode the 8096BH will run in if the CCR is loaded with 0FFH.

CHIP CONFIGURATION REGISTER (CCR)

Configuration information is stored in the Chip Configuration Register (CCR). Four of the bits in the register specify the bus control mode and ready control mode. Two bits also govern the level of ROM/EPROM protection and one bit is NANDed with the BUSWIDTH pin every bus cycle to determine the bus size. The CCR bit map is shown in Figure 10. The functions associated with each bit are described in this section.

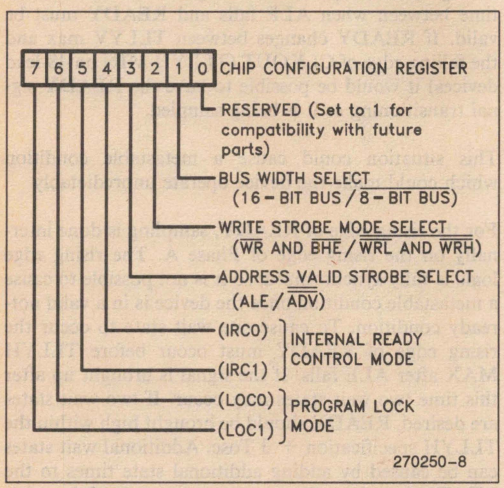


Figure 10. Chip Configuration Register

The CCR is loaded on reset with the Chip Configuration Byte, located at address 2018H. The CCR register is a non-memory mapped location that can only be written to during the reset sequence; once it is loaded it cannot be changed until the next reset occurs. The 8096BH will correctly read this location in every bus mode.

If the $\overline{EA}$ pin is set to a logical 0, the access to 2018H comes from external memory. If $\overline{EA}$ is a logical 1, the access comes from internal ROM/EPROM. If $\overline{EA}$ is +12.5V, the CCR is loaded with a byte from a separate non-memory-mapped location called PCCB (Programming CCB). The Programming mode is described in Section 10 of the Hardware Design chapter.

The CCR is not present on 8X9X devices, but the Chip Configuration Byte at location 2018H should contain the hex value 0FFH to provide future compatibility. 8X9X devices do not access location 2018H on reset.

BUS WIDTH

The 8096BH external bus width can be run-time configured to operate as a standard 16-bit multiplexed address/data bus, or as an 8051 style 16-bit address/8-bit data bus.

During 16-bit bus cycles, Ports 3 and 4 contain the address multiplexed with data using ALE to latch the address. In 8-bit bus cycles, Port 3 is multiplexed address/data while Port 4 is address bits 8 through 15. The address bits on Port 4 are valid throughout an 8-bit bus cycle. Figure 11 shows the two options.

The bus width can be changed each bus cycle and is controlled using bit 1 of the CCR with the BUSWIDTH pin. If either CCR.1 or BUSWIDTH is a 0, external accesses will be over a 16-bit address/8-bit data bus. If both CCR.1 and BUSWIDTH are 1s, external accesses will be over a 16-bit address/16-bit data bus. Internal accesses are always 16-bits wide.

The bus width can be changed every external bus cycle if a 1 was loaded into CCR bit 1 at reset. If this is the case, changing the value of the BUSWIDTH pin at run-time will dynamically select the bus width. For example, the user could feed the INST line into the BUSWIDTH pin, thus causing instruction accesses to be word wide from EPROMs while data accesses are byte wide to and from RAMs. A second example would be to place an inverted version of Address bit 15 on the BUSWIDTH pin. This would make half of external memory word wide, while half is byte wide.

Since BUSWIDTH is sampled after address decoding has had time to occur, even more complex memory maps could be constructed. See the timing specifications for an exact description of BUSWIDTH timings. The bus width will be determined by bit 1 of the CCR alone on 48-pin parts since they do not have a BUSWIDTH pin.

When using an 8-bit bus, some performance degradation is to be expected. On the 8096BH, instruction execution times with an 8-bit bus will slow down if any of three conditions occur. First, word writes to external memory will cause the executing instruction to take two extra state times to complete. Second, word reads from external memory will cause a one state time extension of instruction execution time. Finally, if the prefetch queue is empty when an instruction fetch is requested, instruction execution is lengthened by one state time for each byte that must be externally acquired (worst case is the number of bytes in the instruction minus one.)

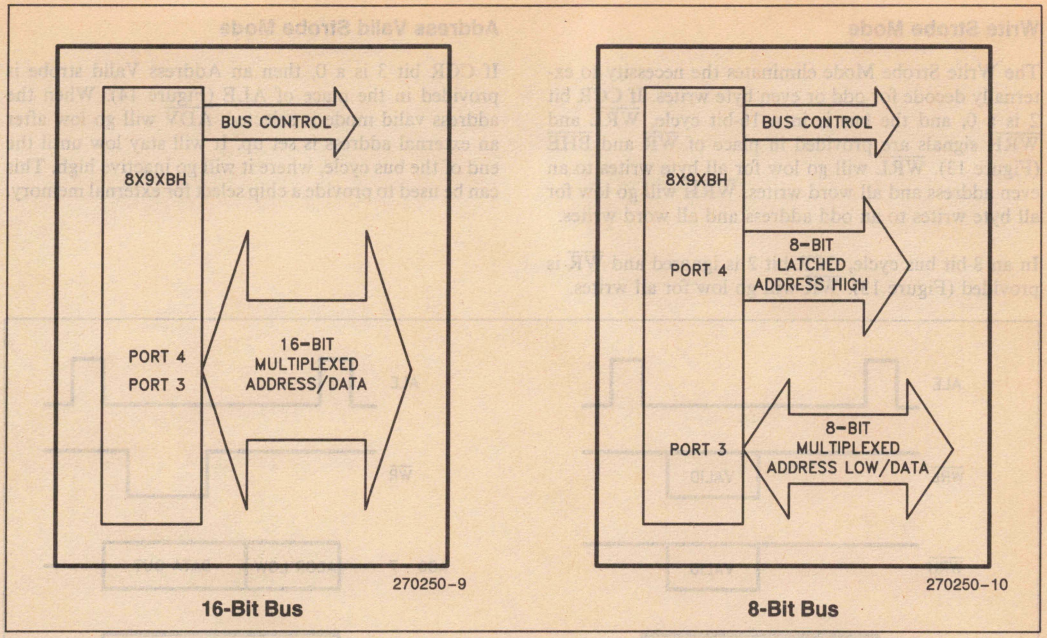


Figure 11. Bus Width Options

BUS CONTROL

Using the CCR, the 8096BH can be made to provide bus control signals of several types. Three control lines have dual functions designed to reduce external hardware. Bits 2 and 3 of the CCR specify the functions performed by these control lines. Figures 12–15 show the signals which can be modified by changing bits in the CCR, all other lines will operate as shown in Figure 9.

Standard Bus Control

If CCR bits 2 and 3 are 1s, then the standard 8096BH control signals $\overline{WR}$, $\overline{BHE}$ and ALE are provided (Figure 12). $\overline{WR}$ will come out for every write. $\overline{BHE}$ will be valid throughout the bus cycle and can be combined with $\overline{WR}$ and address line 0 to form $\overline{WRL}$ and $\overline{WRH}$. ALE will rise as the address starts to come out, and will fall to provide the signal to externally latch the address. *The control signals on 8X9X devices are similar, but not identical to those shown here. Figure 9 shows the 8X9X timings.*

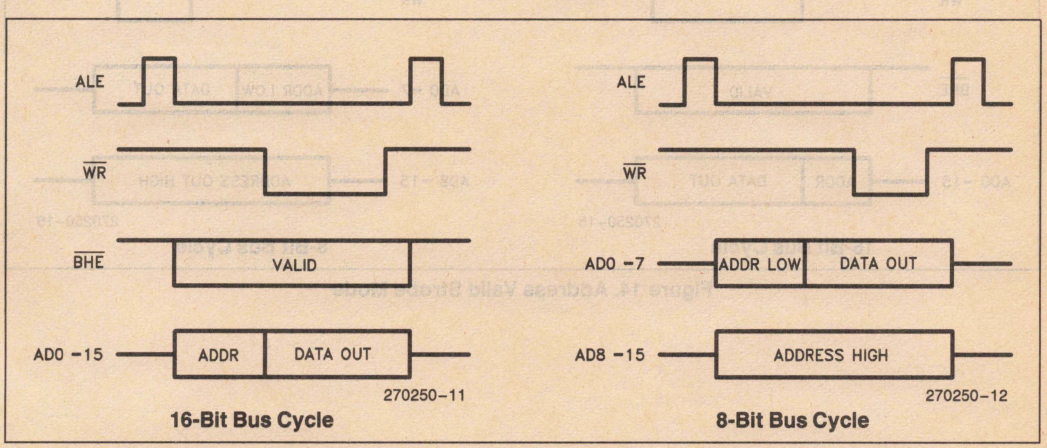


Figure 12. Standard Bus Control

Write Strobe Mode

The Write Strobe Mode eliminates the necessity to externally decode for odd or even byte writes. If CCR bit 2 is a 0, and the bus is in a 16-bit cycle, $\overline{WRL}$ and $\overline{WRH}$ signals are provided in place of $\overline{WR}$ and $\overline{BHE}$ (Figure 13). $\overline{WRL}$ will go low for all byte writes to an even address and all word writes. $\overline{WRH}$ will go low for all byte writes to an odd address and all word writes.

In an 8-bit bus cycle, CCR bit 2 is ignored and $\overline{WR}$ is provided (Figure 13). $\overline{WR}$ will go low for all writes.

Address Valid Strobe Mode

If CCR bit 3 is a 0, then an Address Valid strobe is provided in the place of ALE (Figure 14). When the address valid mode is selected, $\overline{ADV}$ will go low after an external address is set up. It will stay low until the end of the bus cycle, where it will go inactive high. This can be used to provide a chip select for external memory.

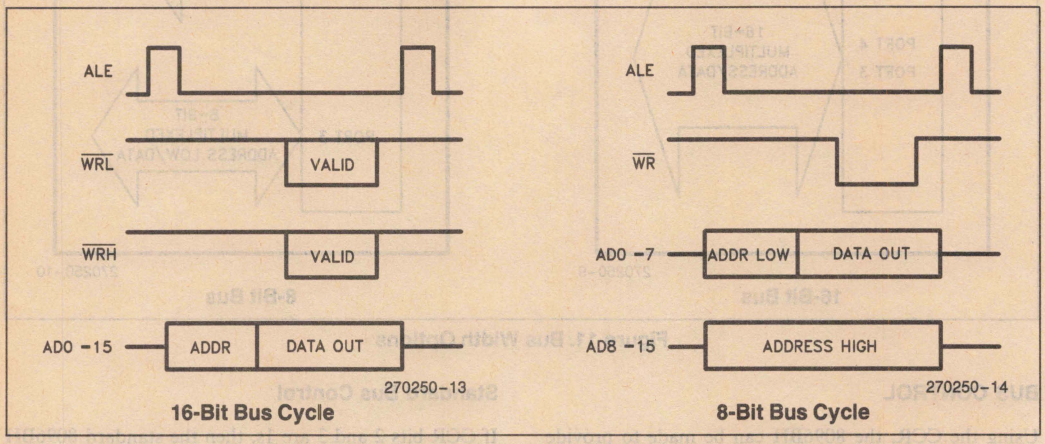


Figure 13. Write Strobe Mode

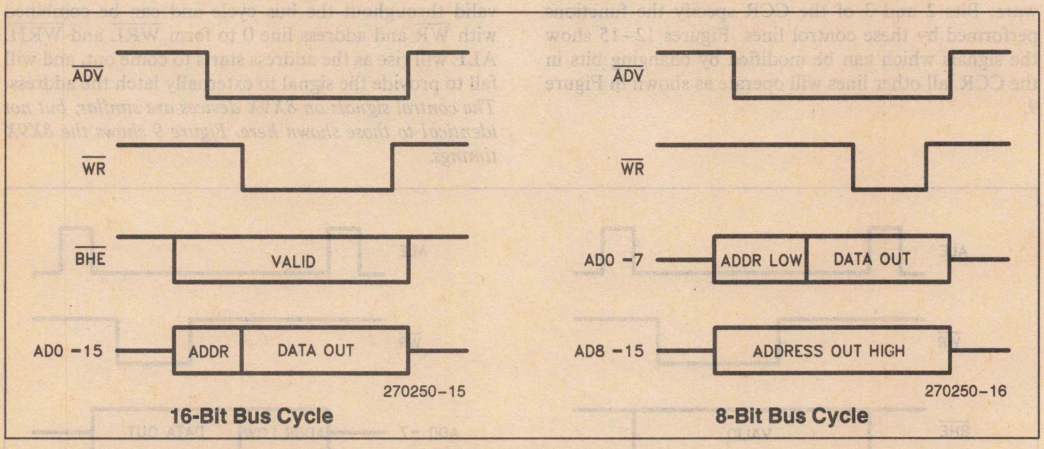


Figure 14. Address Valid Strobe Mode

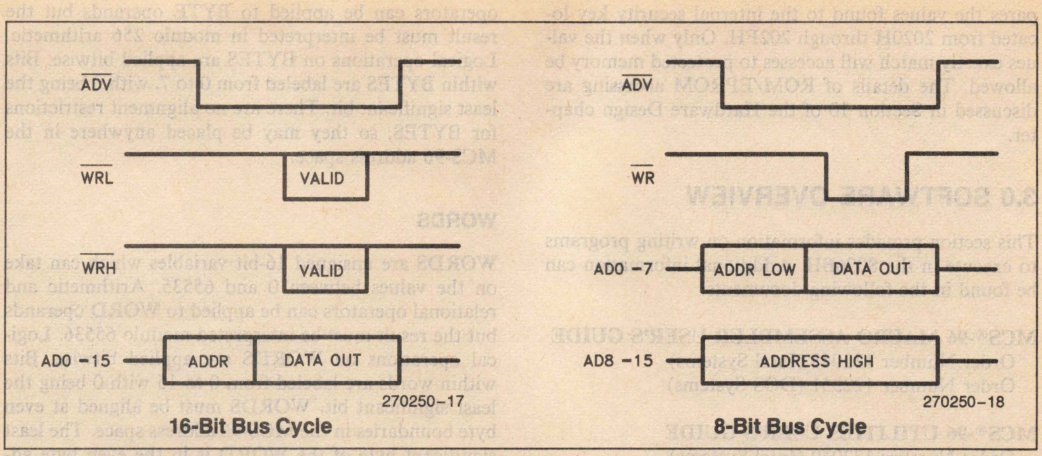


Figure 15. Write Strobe with Address Valid Strobe

Address Valid with Write Strobe

If both CCR bits 2 and 3 are 0s, both the Address Valid strobe and the Write Strobes will be provided for bus control. Figure 15 shows these signals.

READY CONTROL

To simplify ready control, four modes of internal ready control logic have been provided. The modes are chosen by properly configuring bits 4 and 5 of the CCR.

The internal ready control logic can be used to limit the number of wait states that slow devices can insert into the bus cycle. When the READY pin is pulled low, wait states will be inserted into the bus cycle until the READY pin goes high, or the number of wait states equals the number specified by CCR bits 4 and 5, whichever comes first. Table 1 shows the number of wait states that can be selected. Internal Ready control can be disabled by loading 11 into bits 4 and 5 of the CCR.

Table 1. Internal Ready Control

IRC1	IRC0	Description
0	0	Limit to 1 Wait State
0	1	Limit to 2 Wait States
1	0	Limit to 3 Wait States
1	1	Disable Internal Ready Control

This feature provides for simple ready control. For example, every slow memory chip select line could be ORed together and be connected to the READY pin with CCR bits 4 and 5 programmed to give the desired number of wait states to the slow devices.

ROM/EPROM LOCK

Four modes of program memory lock are available on the 839XBH and 879XBH parts. CCR bits 6 and 7 (LOC0, LOC1) select whether internal program memory can be read (or written in EPROM parts) by a program executing from external memory. The modes are shown in Table 2. Internal ROM/EPROM addresses 2020H through 3FFFH are protected from reads while 2000H through 3FFFH are protected from writes, as set by the CCR.

Table 2. Program Lock Modes

LOC1	LOC0	Protection
0	0	Read and Write Protected
0	1	Read Protected
1	0	Write Protected
1	1	No Protection

Only code executing from internal memory can read protected internal memory, while a write protected memory can not be written to, even from internal execution. As a result of 8096BH prefetching of instructions, however, accesses to protected memory are not allowed for instructions located above 3FFAH. If the bus controller receives a request to perform a read of protected memory, the read sequence occurs with indeterminate data being returned to the CPU. Note that the interrupt vectors and the CCR are not protected.

To provide verification and testing when the program lock feature is enabled, the 839XBH and 879XBH verify the security key before programming or test modes are allowed to read from protected memory. Before protected memory can be read, the chip reads external memory locations 4020H through 402FH and com-

pares the values found to the internal security key located from 2020H through 202FH. Only when the values exactly match will accesses to protected memory be allowed. The details of ROM/EPROM accessing are discussed in Section 10 of the Hardware Design chapter.

3.0 SOFTWARE OVERVIEW

This section provides information on writing programs to execute in the 8096BH. Additional information can be found in the following documents:

MCS®-96 MACRO ASSEMBLER USER'S GUIDE
Order Number 122048 (Intel Systems)
Order Number 122351 (DOS Systems)

MCS®-96 UTILITIES USER'S GUIDE
Order Number 122049 (Intel Systems)
Order Number 122356 (DOS Systems)

PL/M-96 USER'S GUIDE
Order Number 122134 (Intel Systems)
Order Number 122361 (DOS Systems)

Throughout this section short sections of code are used to illustrate the operation of the device. For these sections it has been assumed that a set of temporary registers have been predeclared. The names of these registers have been chosen as follows:

AX, BX, CX, and DX are 16-bit registers.

AL is the low byte of AX, AH is the high byte.

BL is the low byte of BX

CL is the low byte of CX

DL is the low byte of DX

These are the same as the names for the general data registers used in the 8086BH. It is important to note, however, that in the 8096, these are not dedicated registers but merely the symbolic names assigned by the programmer to an eight byte region within the onboard register file.

3.1 Operand Types

The MCS®-96 architecture provides support for a variety of data types which are likely to be useful in a control application. In the discussion of these operand types that follows, the names adopted by the PLM-96 programming language will be used where appropriate. To avoid confusion, the name of an operand type will be capitalized. A "BYTE" is an unsigned eight bit variable; a "byte" is an eight bit unit of data of any type.

BYTES

BYTES are unsigned 8-bit variables which can take on the values between 0 and 255. Arithmetic and relational operators can be applied to BYTE operands but the result must be interpreted in modulo 256 arithmetic. Logical operations on BYTES are applied bitwise. Bits within BYTES are labeled from 0 to 7, with 0 being the least significant bit. There are no alignment restrictions for BYTES, so they may be placed anywhere in the MCS-96 address space.

WORDS

WORDS are unsigned 16-bit variables which can take on the values between 0 and 65535. Arithmetic and relational operators can be applied to WORD operands but the result must be interpreted modulo 65536. Logical operations on WORDS are applied bitwise. Bits within words are labeled from 0 to 15 with 0 being the least significant bit. WORDS must be aligned at even byte boundaries in the MCS-96 address space. The least significant byte of the WORD is in the even byte address and the most significant byte is in the next higher (odd) address. The address of a word is the address of its least significant byte. Word operations to odd addresses are not guaranteed to operate in a consistent manner.

SHORT-INTEGERS

SHORT-INTEGERS are 8-bit signed variables which can take on the values between −128 and +127. Arithmetic operations which generate results outside of the range of a SHORT-INTEGER will set the overflow indicators in the program status word. The actual numeric result returned will be the same as the equivalent operation on BYTE variables. There are no alignment restrictions on SHORT-INTEGERS so they may be placed anywhere in the MCS-96 address space.

INTEGERS

INTEGERS are 16-bit signed variables which can take on the values between −32,768 and 32,767. Arithmetic operations which generate results outside of the range of an INTEGER will set the overflow indicators in the program status word. The actual numeric result returned will be the same as the equivalent operation on WORD variables. INTEGERS conform to the same alignment and addressing rules as do WORDS.

BITS

BITS are single-bit operands which can take on the Boolean values of true and false. In addition to the normal support for bits as components of BYTE and WORD operands, the 8096 provides for the direct testing of any bit in the internal register file. The MCS-96 architecture requires that bits be addressed as components of BYTES or WORDS, it does not support the direct addressing of bits that can occur in the MCS-51 architecture.

DOUBLE-WORDS

DOUBLE-WORDS are unsigned 32-bit variables which can take on the values between 0 and 4,294,967,295. The MCS-96 architecture provides direct support for this operand type only for shifts and as the dividend in a 32 by 16 divide and the product of a 16 by 16 multiply. For these operations a DOUBLE-WORD variable must reside in the on-board register file of the 8096 and be aligned at an address which is evenly divisible by 4. A DOUBLE-WORD operand is addressed by the address of its least significant byte. DOUBLE-WORD operations which are not directly supported can be easily implemented with two WORD operations. For consistency with Intel provided software the user should adopt the conventions for addressing DOUBLE-WORD operands which are discussed in Section 3.5.

LONG-INTEGERS

LONG-INTEGERS are 32-bit signed variables which can take on the values between −2,147,483,648 and 2,147,483,647. The MCS-96 architecture provides direct support for this data type only for shifts and as the dividend in a 32 by 16 divide and the product of a 16 by 16 multiply.

LONG-INTEGERS can also be normalized. For these operations a LONG-INTEGER variable must reside in the onboard register file of the 8096 and be aligned at an address which is evenly divisible by 4. A LONG-INTEGER is addressed by the address of its least significant byte.

LONG-INTEGER operations which are not directly supported can be easily implemented with two INTEGER operations. For consistency with Intel provided software, the user should adopt the conventions for addressing LONG operands which are discussed in Section 3.5.

3.2 Operand Addressing

Operands are accessed within the address space of the 8096 with one of six basic addressing modes. Some of the details of how these addressing modes work are hidden by the assembly language. If the programmer is to take full advantage of the architecture, it is important that these details be understood. This section will describe the addressing modes as they are handled by the hardware. At the end of this section the addressing modes will be described as they are seen through the assembly language. The six basic address modes which will be described are termed register-direct, indirect, indirect with auto-increment, immediate, short-indexed, and long-indexed. Several other useful addressing operations can be achieved by combining these basic addressing modes with specific registers such as the ZERO register or the stack pointer.

REGISTER-DIRECT REFERENCES

The register-direct mode is used to directly access a register from the 256 byte on-board register file. The register is selected by an 8-bit field within the instruction and register address and must conform to the alignment rules for the operand type. Depending on the instruction, up to three registers can take part in the calculation.

Examples

```
ADD     AX,BX,CX        ; AX:=BX+CX
MUL     AX,BX           ; AX:=AX*BX
INCB    CL              ; CL:=CL+1
```

INDIRECT REFERENCES

The indirect mode is used to access an operand by placing its address in a WORD variable in the register file. The calculated address must conform to the alignment rules for the operand type. Note that the indirect address can refer to an operand anywhere within the address space of the 8096, including the register file. The register which contains the indirect address is selected by an eight bit field within the instruction. An instruction can contain only one indirect reference and the remaining operands of the instruction (if any) must be register-direct references.

Examples

```
LD      AX,[AX]         ; AX:=MEM_WORD(AX)
ADDB    AL,BL,[CX]      ; AL:=BL+MEM_BYTE(CX)
POP     [AX]            ; MEM_WORD(AX):=MEM_WORD(SP) ; SP:=SP+2
```

INDIRECT WITH AUTO-INCREMENT REFERENCES

This addressing mode is the same as the indirect mode except that the WORD variable which contains the indirect address is incremented *after* it is used to address the operand. If the instruction operates on BYTES or SHORT-INTEGERS the indirect address variable will be incremented by one, if the instruction operates on WORDS or INTEGERS the indirect address variable will be incremented by two.

Examples

```
LD     AX,[BX]+      ; AX:=MEM_WORD(BX) ; BX:=BX+2
ADDB   AL,BL,[CX]+   ; AL:=BL+MEM_BYTE(CX) ; CX:=CX+1
PUSH   [AX]+         ; SP:=SP-2;
                     ;   MEM_WORD(SP):=MEM_WORD(AX)
                     ;   AX:=AX+2
```

IMMEDIATE REFERENCES

This addressing mode allows an operand to be taken directly from a field in the instruction. For operations on BYTE or SHORT-INTEGER operands this field is eight bits wide, for operations on WORD or INTEGER operands the field is 16 bits wide. An instruction can contain only one immediate reference and the remaining operand(s) must be register-direct references.

Examples

```
ADD  AX,#340  ; AX:=AX+340
PUSH #1234H   ; SP:=SP-2; MEM_WORD(SP):=1234H
DIVB AX,#10   ; AL:=AX/10; AH:=AX MOD 10
```

SHORT-INDEXED REFERENCES

In this addressing mode an eight bit field in the instruction selects a WORD variable in the register file which is assumed to contain an address. A second eight bit field in the instruction stream is sign-extended and summed with the WORD variable to form the address of the operand which will take part in the calculation. Since the eight bit field is sign-extended, the effective address can be up to 128 bytes before the address in the WORD variable and up to 127 bytes after it. An instruction can contain only one short-indexed reference and the remaining operand(s) must be register-direct references.

Examples

```
LD     AX,12[BX]     ; AX:=MEM_WORD(BX+12)
MULB   AX,BL,3[CX]   ; AX:=BL*MEM_BYTE(CX+3)
```

LONG-INDEXED REFERENCES

This addressing mode is like the short-indexed mode except that a *16-bit* field is taken from the instruction and added to the WORD variable to form the address of the operand. No sign extension is necessary. An instruction can contain only one long-indexed reference and the remaining operand(s) must be register-direct references.

Examples

```
AND   AX,BX,TABLE[CX]     ; AX:=BX AND MEM_WORD(TABLE+CX)
ST    AX,TABLE[BX]        ; MEM_WORD(TABLE+BX):=AX
ADDB  AL,BL,LOOKUP[CX]    ; AL:=BL+MEM_BYTE(LOOKUP+CX)
```

ZERO REGISTER ADDRESSING

The first two bytes in the register file are fixed at zero by the 8096 hardware. In addition to providing a fixed source of the constant zero for calculations and comparisons, this register can be used as the WORD variable in a long-indexed reference. This combination of register selection and address mode allows any location in memory to be addressed directly.

Examples

```
ADD  AX,1234[0]   ; AX:=AX+MEM_WORD(1234)
POP  5678[0]      ; MEM_WORD(5678):=MEM_WORD(SP)
                  ;  SP:=SP+2
```

STACK POINTER REGISTER ADDRESSING

The system stack pointer in the 8096 can be accessed as register 18H of the internal register file. In addition to providing for convenient manipulation of the stack pointer, this also facilitates the accessing of operands in the stack. The top of the stack, for example, can be accessed by using the stack pointer as the WORD variable in an indirect reference. In a similar fashion, the stack pointer can be used in the short-indexed mode to access data within the stack.

Examples

```
PUSH  [SP]       ; DUPLICATE TOP_OF_STACK
LD    AX,2[SP]   ; AX:=NEXT_TO_TOP
```

ASSEMBLY LANGUAGE ADDRESSING MODES

The 8096 assembly language simplifies the choice of addressing modes to be used in several respects:

Direct Addressing. The assembly language will choose between register-direct addressing and long-indexed with the ZERO register depending on where the operand is in memory. The user can simply refer to an operand by its symbolic name; if the operand is in the register file, a register-direct reference will be used, if the operand is elsewhere in memory, a long-indexed reference will be generated.

Indexed Addressing. The assembly language will choose between short and long indexing depending on the value of the index expression. If the value can be expressed in eight bits then short indexing will be used, if it cannot be expressed in eight bits then long indexing will be used.

The use of these features of the assembly language simplifies the programming task and should be used wherever possible.

3.3 Program Status Word

The program status word (PSW) is a collection of Boolean flags which retain information concerning the state of the user's program. The format of the PSW is shown in Figure 16. The information in the PSW can be broken down into two basic categories; interrupt control and condition flags. The PSW can be saved in the system stack with a single operation (PUSHF) and restored in a like manner (POPF).

15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
Z	N	V	VT	C	—	I	ST	<Interrupt Mask Reg>							

Figure 16. PSW Register

INTERRUPT FLAGS

The lower eight bits of the PSW are used to individually mask the various sources of interrupt to the 8096. A logical '1' in these bit positions enables the servicing of the corresponding interrupt. These mask bits can be accessed as an eight bit byte (INT_MASK—address 8) in the on-board register file. Bit 9 in the PSW is the global interrupt disable. If this bit is cleared then all interrupts will be locked out except for the Non Maskable Interrupt (NMI). Note that the various interrupts are collected in the INT_PENDING register even if they are locked out. Execution of the corresponding service routines will procede according to their priority when they become enabled. Further information on the interrupt structure of the 8096 can be found in Section 4.

CONDITION FLAGS

The remaining bits in the PSW are set as side effects of instruction execution and can be tested by the conditional jump instructions.

Z. The Z (Zero) flag is set to indicate that the operation generated a result equal to zero. For the add-with-carry (ADDC) and subtract-with-borrow (SUBC) operations the Z flag is cleared if the result is non-zero but is never set. These two instructions are normally used in conjunction with the ADD and SUB instructions to perform multiple precision arithmetic. The operation of the Z flag for these instructions leaves it indicating the proper result for the entire multiple precision calculation.

N. The N (Negative) flag is set to indicate that the operation generated a negative result. Note that the N flag will be set to the algebraically correct state even if the calculation overflows. When the NEGB instruction is performed on a byte register containing 80H, or the NEG instruction is performed on a word register containing 8000H, the N flag is set.

V. The V (overflow) flag is set to indicate that the operation generated a result which is outside the range that can be expressed in the destination data type. For the SHL, SHLB and SHLL instructions, the V flag will be set if the most significant bit of the operand changes at any time during the shift.

VT. The VT (oVerflow Trap) flag is set whenever the V flag is set but can only be cleared by an instruction which explicitly operates on it such as the CLRVT or JVT instructions. The operation of the VT flag allows for the testing for a possible overflow condition at the end of a sequence of related arithmetic operations. This is normally more efficient than testing the V flag after each instruction.

C. The C (Carry) flag is set to indicate the state of the arithmetic carry from the most significant bit of the ALU for an arithmetic operation or the state of the last bit shifted out of the operand for a shift. Arithmetic Borrow after a subtract operation is the complement of the C flag (i.e. if the operation generated a borrow then C = 0).

ST. The ST (STicky bit) flag is set to indicate that during a right shift a 1 has been shifted first into the C flag and then been shifted out. The ST flag is undefined after a multiply operation. The ST flag can be used along with the C flag to control rounding after a right shift. Consider multiplying two eight bit quantities and then scaling the result down to 12 bits:

```
MULUB   AX,CL,DL    ;AX:=CL*DL
SHR     AX,#4       ;Shift right 4 places
```

If the C flag is set after the shift, it indicates that the bits shifted off the end of the operand were greater-than or equal-to one half the least significant bit (LSB) of the result. If the C flag is clear after the shift, it indicates that the bits shifted off the end of the operand were less than half the LSB of the result. Without the ST flag, the rounding decision must be made on the basis of this information alone. (Normally the result would be rounded up if the C flag is set.) The ST flag allows a finer resolution in the rounding decision:

C ST	Value of the Bits Shifted Off
0 0	Value = 0
0 1	0 < Value < ½ LSB
1 0	Value = ½ LSB
1 1	Value > ½ LSB

Figure 17. Rounding Alternatives

Imprecise rounding can be a major source of error in a numerical calculation; use of the ST flag improves the options available to the programmer.

3.4 Instruction Set

The MCS-96 instruction set contains a full set of arithmetic and logical operations for the 8-bit data types BYTE and SHORT INTEGER and for the 16-bit data types WORD and INTEGER. The DOUBLE-WORD and LONG data types (32 bits) are supported for the products of 16 by 16 multiplies and the dividends of 32

by 16 divides and for shift operations. The remaining operations on 32-bit variables can be implemented by combinations of 16-bit operations. As an example the sequence:

```
ADD     AX,CX
ADDC    BX,DX
```

performs a 32-bit addition, and the sequence

```
SUB     AX,CX
SUBC    BX,DX
```

performs a 32-bit subtraction. Operations on REAL (i.e. floating point) variables are not supported directly by the hardware but are supported by the floating point library for the 8096 (FPAL-96) which implements a single precision subset of the proposed IEEE standard for floating point operations. The performance of this software is significantly improved by the 8096 NORML instruction which normalizes a 32-bit variable and by the existence of the ST flag in the PSW.

In addition to the operations on the various data types, the 8096 supports conversions between these types. LDBZE (load byte zero extended) converts a BYTE to a WORD and LDBSE (load byte sign extended) converts a SHORT-INTEGER into an INTEGER. WORDS can be converted to DOUBLE-WORDS by simply clearing the upper WORD of the DOUBLE-WORD (CLR) and INTEGERS can be converted to LONGS with the EXT (sign extend) instruction.

The MCS-96 instructions for addition, subtraction, and comparison do not distinguish between unsigned words and signed integers. Conditional jumps are provided to allow the user to treat the results of these operations as either signed or unsigned quantities. As an example, the CMPB (compare byte) instruction is used to compare both signed and unsigned eight bit quantities. A JH (jump if higher) could be used following the compare if unsigned operands were involved or a JGT (jump if greater-than) if signed operands were involved.

Table 3 summarizes the operation of each of the instructions. Complete descriptions of each instruction and its timings can be found in the Instruction Set chapter. A summary of instruction opcodes and timing is included in the quick reference section at the end of this chapter. Examples of using the instruction set of the MCS-96 family can be found in Application Note AP-248, "Using the 8096", included in this handbook.

Table 3. Instruction Summary

Mnemonic	Operands	Operation (Note 1)	Flags						Notes
			Z	N	C	V	VT	ST	
ADD/ADDB	2	D ← D + A	↕	↕	↕	↕	↑	—	
ADD/ADDB	3	D ← B + A	↕	↕	↕	↕	↑	—	
ADDC/ADDCB	2	D ← D + A + C	↓	↕	↕	↕	↑	—	
SUB/SUBB	2	D ← D − A	↕	↕	↕	↕	↑	—	
SUB/SUBB	3	D ← B − A	↕	↕	↕	↕	↑	—	
SUBC/SUBCB	2	D ← D − A + C − 1	↓	↕	↕	↕	↑	—	
CMP/CMPB	2	D − A	↕	↕	↕	↕	↑	—	
MUL/MULU	2	D, D + 2 ← D * A	—	—	—	—	—	?	2
MUL/MULU	3	D, D + 2 ← B * A	—	—	—	—	—	?	2
MULB/MULUB	2	D, D + 1 ← D * A	—	—	—	—	—	?	3
MULB/MULUB	3	D, D + 1 ← B * A	—	—	—	—	—	?	3
DIVU	2	D ← (D, D + 2)/A, D + 2 ← remainder	—	—	—	↕	↑	—	2
DIVUB	2	D ← (D, D + 1)/A, D + 1 ← remainder	—	—	—	↕	↑	—	3
DIV	2	D ← (D, D + 2)/A, D + 2 ← remainder	—	—	—	?	↑	—	
DIVB	2	D ← (D, D + 1)/A, D + 1 ← remainder	—	—	—	?	↑	—	
AND/ANDB	2	D ← D and A	↕	↕	0	0	—	—	
AND/ANDB	3	D ← B and A	↕	↕	0	0	—	—	
OR/ORB	2	D ← D or A	↕	↕	0	0	—	—	
XOR/XORB	2	D ← D (excl. or) A	↕	↕	0	0	—	—	
LD/LDB	2	D ← A	—	—	—	—	—	—	
ST/STB	2	A ← D	—	—	—	—	—	—	
LDBSE	2	D ← A; D + 1 ← SIGN(A)	—	—	—	—	—	—	3, 4
LDBZE	2	D ← A; D + 1 ← 0	—	—	—	—	—	—	3, 4
PUSH	1	SP ← SP − 2; (SP) ← A	—	—	—	—	—	—	
POP	1	A ← (SP); SP ← SP + 2	—	—	—	—	—	—	
PUSHF	0	SP ← SP − 2; (SP) ← PSW; PSW ← 0000H I ← 0	0	0	0	0	0	0	
POPF	0	PSW ← (SP); SP ← SP + 2; I ← ↕	↕	↕	↕	↕	↕	↕	
SJMP	1	PC ← PC + 11-bit offset	—	—	—	—	—	—	5
LJMP	1	PC ← PC + 16-bit offset	—	—	—	—	—	—	5
BR [indirect]	1	PC ← (A)	—	—	—	—	—	—	
SCALL	1	SP ← SP − 2; (SP) ← PC; PC ← PC + 11-bit offset	—	—	—	—	—	—	5
LCALL	1	SP ← SP − 2; (SP) ← PC; PC ← PC + 16-bit offset	—	—	—	—	—	—	5
RET	0	PC ← (SP); SP ← SP + 2	—	—	—	—	—	—	
J (conditional)	1	PC ← PC + 8-bit offset (if taken)	—	—	—	—	—	—	5
JC	1	Jump if C = 1	—	—	—	—	—	—	5
JNC	1	Jump if C = 0	—	—	—	—	—	—	5
JE	1	Jump if Z = 1	—	—	—	—	—	—	5

NOTES:
1. If the mnemonic ends in "B", a byte operation is performed, otherwise a word operation is done. Operands D, B, and A must conform to the alignment rules for the required operand type. D and B are locations in the register file; A can be located anywhere in memory.
2. D, D + 2 are consecutive WORDS in memory; D is DOUBLE-WORD aligned.
3. D, D + 1 are consecutive BYTES in memory; D is WORD aligned.
4. Changes a byte to a word.
5. Offset is a 2's complement number.

Table 3. Instruction Summary (Continued)

Mnemonic	Operands	Operation (Note 1)	Flags						Notes
			Z	N	C	V	VT	ST	
JNE	1	Jump if Z = 0	—	—	—	—	—	—	5
JGE	1	Jump if N = 0	—	—	—	—	—	—	5
JLT	1	Jump if N = 1	—	—	—	—	—	—	5
JGT	1	Jump if N = 0 and Z = 0	—	—	—	—	—	—	5
JLE	1	Jump if N = 1 or Z = 1	—	—	—	—	—	—	5
JH	1	Jump if C = 1 and Z = 0	—	—	—	—	—	—	5
JNH	1	Jump if C = 0 or Z = 1	—	—	—	—	—	—	5
JV	1	Jump if V = 1	—	—	—	—	—	—	5
JNV	1	Jump if V = 0	—	—	—	—	—	—	5
JVT	1	Jump if VT = 1; Clear VT	—	—	—	—	0	—	5
JNVT	1	Jump if VT = 0; Clear VT	—	—	—	—	0	—	5
JST	1	Jump if ST = 1	—	—	—	—	—	—	5
JNST	1	Jump if ST = 0	—	—	—	—	—	—	5
JBS	3	Jump if Specified Bit = 1	—	—	—	—	—	—	5, 6
JBC	3	Jump if Specified Bit = 0	—	—	—	—	—	—	5, 6
DJNZ	1	D ← D − 1; if D ≠ 0 then PC ← PC + 8-bit offset	—	—	—	—	—	—	5
DEC/DECB	1	D ← D − 1	↙	↙	↙	↙	↑	—	
NEG/NEGB	1	D ← 0 − D	↙	↙	↙	↙	↑	—	
INC/INCB	1	D ← D + 1	↙	↙	↙	↙	↑	—	
EXT	1	D ← D; D + 2 ← Sign (D)	↙	↙	0	0	—	—	2
EXTB	1	D ← D; D + 1 ← Sign(D)	↙	↙	0	0	—	—	3
NOT/NOTB	1	D ← Logical Not (D)	↙	↙	0	0	—	—	
CLR/CLRB	1	D ← 0	1	0	0	0	—	—	
SHL/SHLB/SHLL	2	C ← msb — — — — — — lsb ← 0	↙	?	↙	↙	↑	—	7
SHR/SHRB/SHRL	2	0 → msb — — — — — — lsb → C	↙	?	↙	0	—	↙	7
SHRA/SHRAB/SHRAL	2	msb → msb — — — — — — lsb → C	↙	↙	↙	0	—	↙	7
SETC	0	C ← 1	—	—	1	—	—	—	
CLRC	0	C ← 0	—	—	0	—	—	—	
CLRVT	0	VT ← 0	—	—	—	—	0	—	
RST	0	PC ← 2080H	0	0	0	0	0	0	8
DI	0	Disable All Interrupts (I ← 0)	—	—	—	—	—	—	
EI	0	Enable All Interrupts (I ← 1)	—	—	—	—	—	—	
NOP	0	PC ← PC + 1	—	—	—	—	—	—	
SKIP	0	PC ← PC + 2	—	—	—	—	—	—	
NORML	2	Left shift till msb = 1; D ← shift count	↙	?	0	—	—	—	7
TRAP	0	SP ← SP − 2; (SP) ← PC PC ← (2010H)	—	—	—	—	—	—	9

NOTES:
1. If the mnemonic ends in "B", a byte operation is performed, otherwise a word operation is done. Operands D, B and A must conform to the alignment rules for the required operand type. D and B are locations in the register file; A can be located anywhere in memory.
5. Offset is a 2's complement number.
6. Specified bit is one of the 2048 bits in the register file.
7. The "L" (Long) suffix indicates double-word operation.
8. Initiates a Reset by pulling RESET low. Software should re-initialize all the necessary registers with code starting at 2080H.
9. The assembler will not accept this mnemonic.

3.5 Software Standards and Conventions

For a software project of any size it is a good idea to modularize the program and to establish standards which control the communication between these modules. The nature of these standards will vary with the needs of the final application. A common component of all of these standards, however, must be the mechanism for passing parameters to procedures and returning results from procedures. In the absence of some overriding consideration which prevents their use, it is suggested that the user conform to the conventions adopted by the PLM-96 programming language for procedure linkage. It is a very usable standard for both the assembly language and PLM-96 environment and it offers compatibility between these environments. Another advantage is that it allows the user access to the same floating point arithmetics library that PLM-96 uses to operate on REAL variables.

REGISTER UTILIZATION

The MCS-96 architecture provides a 256 byte register file. Some of these registers are used to control register-mapped I/O devices and for other special functions such as the ZERO register and the stack pointer. The remaining bytes in the register file, some 230 of them, are available for allocation by the programmer. If these registers are to be used effectively, some overall strategy for their allocation must be adopted. PLM-96 adopts the simple and effective strategy of allocating the eight bytes between addresses 1CH and 23H as temporary storage. The starting address of this region is called PLMREG. The remaining area in the register file is treated as a segment of memory which is allocated as required.

ADDRESSING 32-BIT OPERANDS

These operands are formed from two adjacent 16-bit words in memory. The least significant word of the double word is always in lower address, even when the data is in the stack (which means that the most significant word must be pushed into the stack first). A double word is addressed by the address of its least significant byte. Note that the hardware supports some operations on double words (e.g. normalize and divide). For these operations the double word must be in the internal register file and must have an address which is evenly divisible by four.

SUBROUTINE LINKAGE

Parameters are passed to subroutines in the stack. Parameters are pushed into the stack in the order that they are encountered in the scanning of the source text. Eight-bit parameters (BYTES or SHORT-INTEGERS) are pushed into the stack with the high order byte undefined. Thirty-two bit parameters (LONG-INTEGERS, DOUBLE-WORDS, and REALS) are pushed into the stack as two 16-bit values; the most significant half of the parameter is pushed into the stack first.

As an example, consider the following PLM-96 procedure:

```
example_procedure: PROCEDURE
(param1,param2,param3);
  DECLARE param1 BYTE,
          param2 DWORD,
          param3 WORD;
```

When this procedure is entered at run time the stack will contain the parameters in the following order:

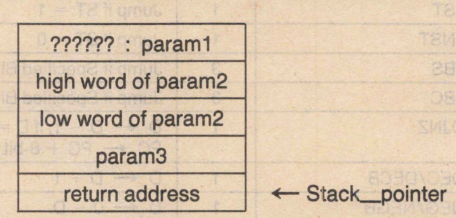


Figure 18. Stack Image

If a procedure returns a value to the calling code (as opposed to modifying more global variables) then the result is returned in the variable PLMREG. PLMREG is viewed as either an 8-, 16- or 32-bit variable depending on the type of the procedure.

The standard calling convention adopted by PLM-96 has several key features:

a) Procedures can always assume that the eight bytes of register file memory starting at PLMREG can be used as temporaries within the body of the procedure.

b) Code which calls a procedure must assume that the eight bytes of register file memory starting at PLMREG are modified by the procedure.

c) The Program Status Word (PSW—see Section 3.3) is not saved and restored by procedures so the calling code must assume that the condition flags (Z, N, V, VT, C, and ST) are modified by the procedure.

d) Function results from procedures are always returned in the variable PLMREG.

PLM-96 allows the definition of INTERRUPT procedures which are executed when a predefined interrupt occurs. These procedures do not conform to the rules of a normal procedure. Parameters cannot be passed to these procedures and they cannot return results. Since they can execute essentially at any time (hence the term interrupt), these procedures must save the PSW and PLMREG when they are entered and restore these values before they exit.

4.0 INTERRUPT STRUCTURE

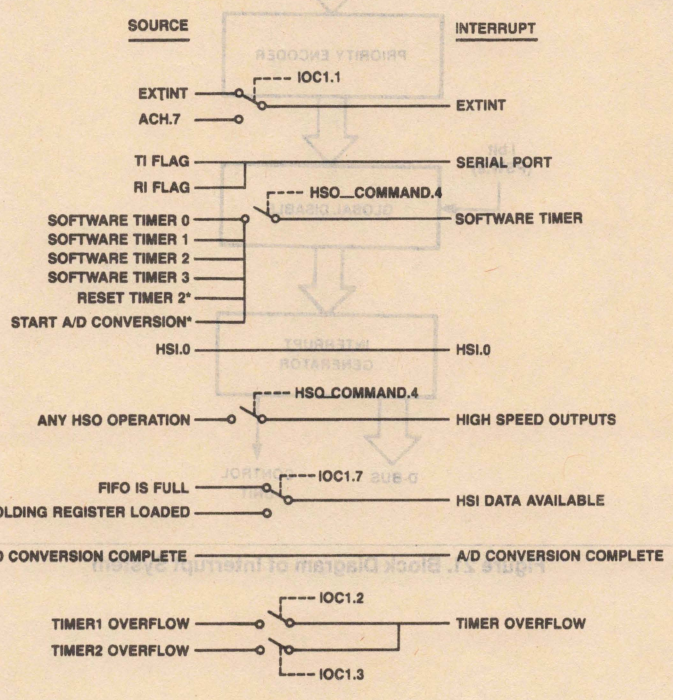
There are 21 sources of interrupts on the 8096BH. These sources are gathered into 8 interrupt types as indicated in Figure 19. The I/O control registers which control some of the sources are indicated in the figure. Each of the eight types of interrupts has its own interrupt vector as listed in Figure 20. In addition to the 8 standard interrupts, there is a TRAP instruction which acts as a software generated interrupt. This instruction is not currently supported by the MCS-96 Assembler and is reserved for use in Intel development systems.

The programmer must initialize the interrupt vector table with the starting address of the appropriate interrupt service routine. It is suggested that any unused interrupts be vectored to an error handling routine.

Three registers control the operation of the interrupt system: Interrupt Pending, Interrupt Mask, and the PSW which contains a global disable bit. A block diagram of the system is shown in Figure 21. The transition detector looks for 0 to 1 transitions on any of the sources. External sources have a maximum transition speed of one edge every state time. If this is exceeded the interrupt may not be detected.

Vector	Vector Location (High Byte)	Vector Location (Low Byte)	Priority
Software	2011H	2010H	Not Applicable
Extint	200FH	200EH	7 (Highest)
Serial Port	200DH	200CH	6
Software Timers	200BH	200AH	5
HSI.0	2009H	2008H	4
High Speed Outputs	2007H	2006H	3
HSI Data Available	2005H	2004H	2
A/D Conversion Complete	2003H	2002H	1
Timer Overflow	2001H	2000H	0 (Lowest)

Figure 20. Interrupt Vector Locations



NOTE:
*Only when initiated by the HSO unit.

Figure 19. All Possible Interrupt Sources

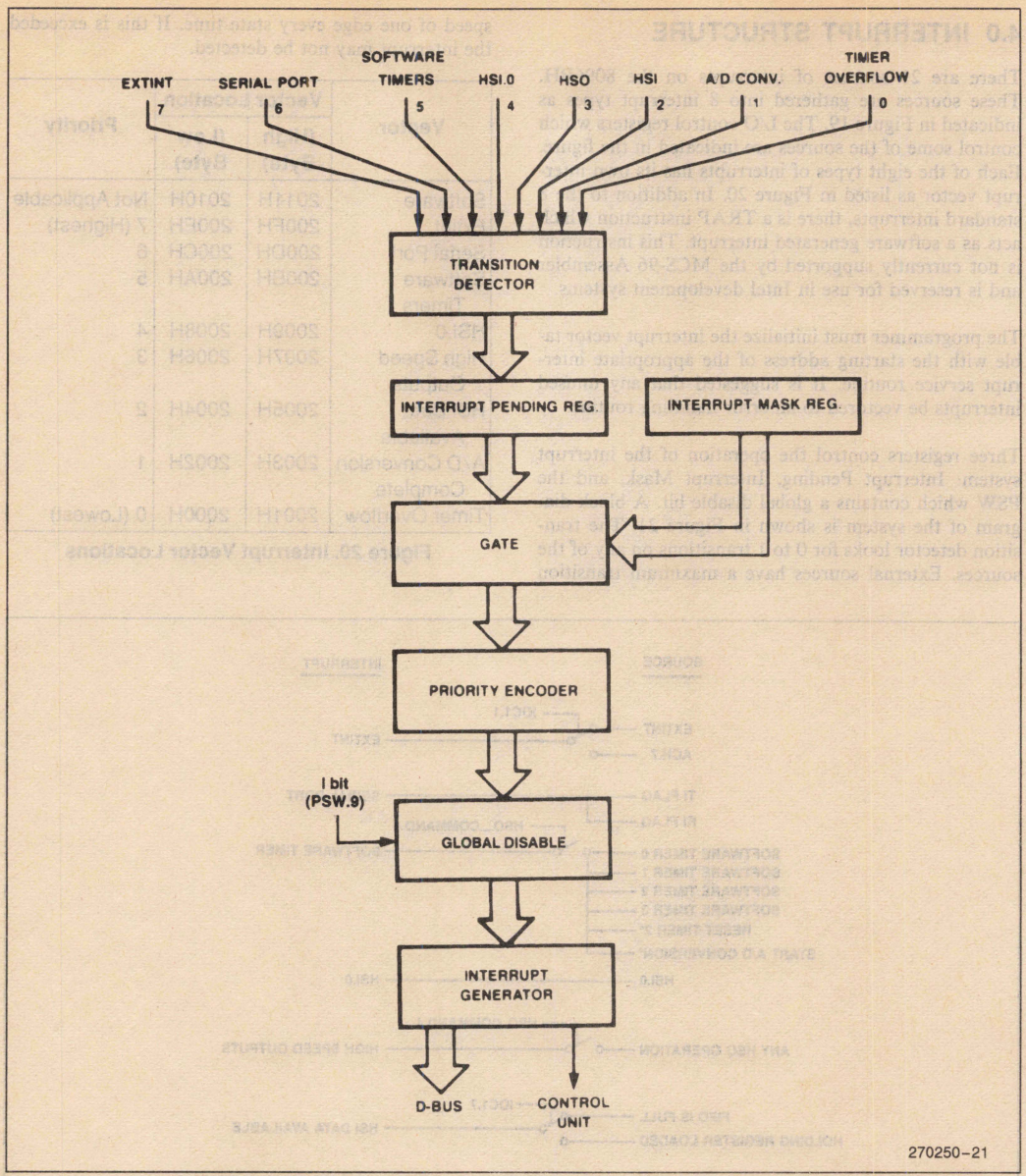


Figure 21. Block Diagram of Interrupt System

4.1 Interrupt Control

Interrupt Pending Register

When the hardware detects one of the eight interrupts it sets the corresponding bit in the pending interrupt register (INT_PENDING-09H). When the interrupt vector is taken, the pending bit is cleared. This register, the format of which is shown in Figure 22, can be read or modified as a byte register. It can be read to determine which of the interrupts are pending at any given time or modified to either clear pending interrupts or generate interrupts under software control. Any software which modifies the INT_PENDING register should ensure that the entire operation is indivisible. The easiest way to do this is to use the logical instructions in the two or three operand format, for example:

```
ANDB   INT_PENDING,#11111101B
         ; Clears the A/D Interrupt
ORB    INT_PENDING,#00000010B
         ; Sets the A/D Interrupt
```

Caution must be used when writing to the pending register to clear interrupts. If the interrupt has already been acknowledged when the bit is cleared, a 4 state time "partial" interrupt cycle will occur. This is because the 8096BH will have to fetch the next instruction of the normal instruction flow, instead of proceeding with the interrupt processing as it was going to. The effect on the program will be essentially that of an extra NOP. This can be prevented by clearing the bits using a 2 operand immediate logical, as the 8096BH holds off acknowledging interrupts during these "read/modify/write" instructions.

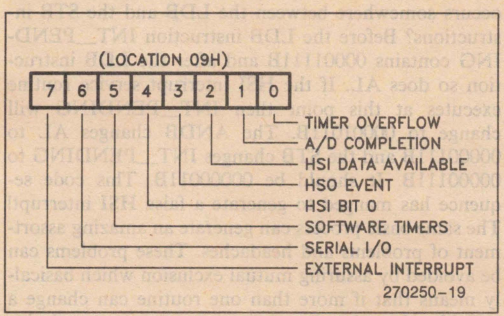


Figure 22. Interrupt Pending Register

Interrupt Mask Register

Individual interrupts can be enabled or disabled by setting or clearing bits in the interrupt mask register (INT_MASK-08H). The format of this register is the same as that of the Interrupt Pending Register shown in Figure 22.

The INT_MASK register can be read or written as byte register. A one in any bit position will enable the corresponding interrupt source and a zero will disable the source. The hardware will save any interrupts that occur by setting bits in the pending register, even if the interrupt mask bit is cleared. The INT_MASK register also can be accessed as the lower eight bits of the PSW so the PUSHF and POPF instructions save and restore the INT_MASK register as well as the global interrupt lockout and the arithmetic flags.

GLOBAL DISABLE

The processing of all interrupts can be disabled by clearing the I bit in the PSW. Setting the I bit will enable interrupts that have mask register bits which are set. The I bit is controlled by the EI (Enable Interrupts) and DI (Disable Interrupts) instructions. Note that the I bit only controls the actual servicing of interrupts. Interrupts that occur during periods of lockout will be held in the pending register and serviced on a prioritized basis when the lockout period ends.

4.2 Interrupt Priorities

The priority encoder looks at all of the interrupts which are both pending and enabled, and selects the one with the highest priority. The priorities are shown in Figure 20 (7 is highest, 0 is lowest). The interrupt generator then forces a call to the location in the indicated vector location. This location would be the starting location of the Interrupt Service Routine (ISR).

This priority selection controls the order in which pending interrupts are passed to the software via interrupt calls. The software can then implement its own priority structure by controlling the mask register (INT_MASK). To see how this is done, consider the case of a serial I/O service routine which must run at a priority level which is lower than the HSI data available interrupt but higher than any other source. The "preamble" and exit code for this interrupt service routine would look like this:

```
serial_io_isr:
 PUSHF          ; Save the PSW
                  (Includes INT_MASK)
 LDB    INT_MASK,#00000100B
 EI             ; Enable interrupts again
 ;
 ;
 ;
 ;     Service the interrupt
 ;
 ;
 ;
 POPF           ; Restore the PSW
 RET
```

Note that location 200CH in the interrupt vector table would have to be loaded with the value of the label serial_io_isr and the interrupt be enabled for this routine to execute.

There is an interesting chain of instruction side-effects which makes this (or any other) 8096 interrupt service routine execute properly:

a) After the hardware decides to process an interrupt, it generates and executes a special interrupt-call instruction, which pushes the current program counter onto the stack and then loads the program counter with the contents of the vector table entry corresponding to the interrupt. The hardware will not allow another interrupt to be serviced immediately following the interrupt-call. This guarantees that once the interrupt-call starts, the first instruction of the interrupt service routine will execute.

b) The PUSHF instruction, which is now guaranteed to execute, saves the PSW in the stack and then clears the PSW. The PSW contains, in addition to the arithmetic flags, the INT_MASK register and the global disable flag (I). The hardware will not allow an interrupt following a PUSHF instruction and, by the time the LD instruction starts, all of the interrupt enable flags will be cleared. Now there is guaranteed execution of the LD INT_MASK instruction.

c) The LD INT_MASK instruction enables those interrupts that the programmer chooses to allow to interrupt the serial I/O interrupt service routine. In this example only the HSI data available interrupt will be allowed to do this but any interrupt or combination of interrupts could be enabled at this point, even the serial interrupt. It is the loading of the INT_MASK register which allows the software to establish its own priorities for interrupt servicing independently from those that the hardware enforces.

d) The EI instruction reenables the processing of interrupts.

e) The actual interrupt service routine executes within the priority structure established by the software.

f) At the end of the service routine the POPF instruction restores the PSW to its state when the interrupt-call occurred. The hardware will not allow interrupts to be processed following a POPF instruction so the execution of the last instruction (RET) is guaranteed before further interrupts can occur. The reason that this RET instruction must be protected in this fashion is that it is quite likely that the POPF instruction will reenable an interrupt which is already pending. If this interrupt were serviced before the RET instruction, then the return address to the code that was executing when the original interrupt occurred would be left on the stack. While this does not present a problem to the program flow, it could result in a stack overflow if interrupts are occurring at a high frequency. The POPF instruction also pops the INT_MASK register (part of the PSW), so any changes made to this register during a routine which ends with a POPF will be lost.

Notice that the "preamble" and exit code for the interrupt service routine does not include any code for saving or restoring registers. This is because it has been assumed that the interrupt service routine has been allocated its own private set of registers from the on-board register file. The availability of some 230 bytes of register storage makes this quite practical.

4.3 Critical Regions

Interrupt service routines must share some data with other routines. Whenever the programmer is coding those sections of code which access these shared pieces of data, great care must be taken to ensure that the integrity of the data is maintained. Consider clearing a bit in the interrupt pending register as part of a non-interrupt routine:

```
LDB     AL,INT_PENDING
ANDB    AL,#bit_mask
STB     AL,INT_PENDING
```

This code works if no other routines are operating concurrently, but will cause occasional but serious problems if used in a concurrent environment. (All programs which make use of interrupts must be considered to be part of a concurrent environment.) To demonstrate this problem, assume that the INT_PENDING register contains 00001111B and bit 3 (HSO event interrupt pending) is to be reset. The code does work for this data pattern but what happens if an HSI interrupt occurs somewhere between the LDB and the STB instructions? Before the LDB instruction INT_PENDING contains 00001111B and after the LDB instruction so does AL. If the HSI interrupt service routine executes at this point then INT_PENDING will change to 00001011B. The ANDB changes AL to 00000111B and the STB changes INT_PENDING to 00000111B. It should be 00000011B. This code sequence has manged to generate a false HSI interrupt! The same basic process can generate an amazing assortment of problems and headaches. These problems can be avoided by assuring mutual exclusion which basically means that if more than one routine can change a variable, then the programmer must ensure exclusive access to the variable during the entire operation on the variable.

In many cases the instruction set of the 8096 allows the variable to be modified with a single instruction. The code in the above example can be implemented with a single instruction.

```
ANDB    INT_PENDING,#bit_mask
```

Instructions are indivisible so mutual exclusion is ensured in this case. Changes to the INT_PENDING register must be made as a single instruction, since bits can be changed in this register even if interrupts are disabled. Depending on system configurations, several other SFRs might also need to be changed in a single instruction for the same reason.

When variables must be modified without interruption, and a single instruction can not be used, the programmer must create what is termed a critical region in which it is safe to modify the variable. One way to do this is to simply disable interrupts with a DI instruction, perform the modification, and then re-enable interrupts with an EI instruction. The problem with this approach is that it leaves the interrupts enabled even if they were not enabled at the start. A better solution is to enter the critical region with a PUSHF instruction which saves the PSW and also clears the interrupt enable flags. The region can then be terminated with a POPF instruction which returns the interrupt enable to the state it was in before the code sequence. It should be noted that some system configurations might require more protection to form a critical region. An example is a system in which more than one processor has access to a common resource such as memory or external I/O devices.

4.4 Interrupt Timing

Interrupts are not always acknowledged immediately. If the interrupt signal does not occur prior to 4 state-times before the end of an instruction, the interrupt will not be acknowledged until after the next instruction has been executed. This is because an instruction is fetched and prepared for execution a few state times before it is actually executed.

There are 6 instructions which always inhibit interrupts from being acknowledged until after the next instruction has been executed. These instructions are:

EI, DI — Enable and Disable Interrupts

POPF, PUSHF— Pop and Push Flags

SIGND — Prefix to perform signed multiply and divide (Note that this is not an ASM-96 Mnemonic, but is used for signed multiply and divide)

TRAP — Software interrupt

When an interrupt is acknowledged, the interrupt pending bit is cleared, and a call is forced to the location indicated by the specified interrupt vector. This call occurs after the completion of the instruction in process, except as noted above. The procedure of getting the vector and forcing the call requires 21 state times. If the stack is in external RAM an additional 3 state times are required.

The maximum number of state times required from the time an interrupt is generated (not acknowledged) until the 8096 begins executing code at the desired location is the time of the longest instruction, NORML (Normalize — 43 state times), plus the 4 state times prior to the end of the previous instruction, plus the response time (21 to 24 state times). Therefore, the maximum response time is 71 (43 + 4 + 24) state times. This does not include the 12 state times required for PUSHF if it is used as the first instruction in the interrupt routine or additional latency caused by having the interrupt masked or disabled.

Interrupt latency time can be reduced by careful selection of instructions in areas of code where interrupts are expected. Using 'EI' followed immediately by a long instruction (e.g. MUL, NORML, etc.) will increase the maximum latency by 4 state times, as an interrupt cannot occur between EI and the instruction following EI. The "DI", "PUSHF", "POPF" and "TRAP" instructions will also cause the same situation. Typically the PUSHF, POPF and TRAP instructions would only effect latency when one interrupt routine is already in process, as these instructions are seldom used at other times.

5.0 TIMERS

Two 16-bit timers are available for use on the 8096. The first is designated "Timer 1", the second, "Timer 2". Timer 1 is used to synchronize events to real time, while Timer 2 can be clocked externally and synchronizes events to external occurrences.

5.1 Timer 1

Timer 1 is clocked once every eight state times and can be cleared only by executing a reset. The only other way to change its value is by writing to 000CH but this is a test mode which sets both timers to 0FFFXH and should not be used in programs.

5.2 Timer 2

Timer 2 can be incremented by transitions (one count each transition, rising *and* falling) on either T2CLK or HSI.1. The multiple functionality of the timer is determined by the state of I/O Control Register 0, bit 7 (IOC0.7). To ensure that all CAM entries are checked

each count of Timer 2, the maximum transition speed is limited to once per eight state times. Timer 2 can be cleared by: executing a reset, by setting IOC0.1, by triggering HSO channel 0EH, or by pulling T2RST or HSI.0 high. The HSO and CAM are described in Section 7 and 8. IOC0.3 and ICO0.5 control the resetting of Timer 2. Figure 23 shows the different ways of manipulating Timer 2.

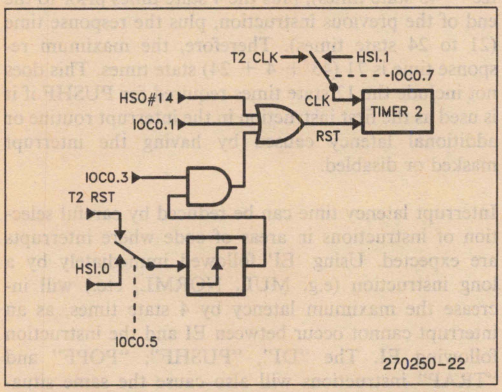


Figure 23. Timer 2 Clock and Reset Options

5.3 Timer Interrupts

Both Timer 1 and Timer 2 can be used to trigger a timer overflow interrupt and set a flag in the I/O Status Register 1 (IOS1). The interrupts are controlled by IOC1.2 and IOC1.3 respectively. The flags are set in IOS1.5 and IOS1.4, respectively.

Caution must be used when examining the flags, as any access (including Compare and Jump on Bit) of IOS1 clears bits 0 through 5 including the software timer flags. It is, therefore, recommended to write the byte to a temporary register before testing bits. The general enabling and disabling of the timer interrupts are controlled by the Interrupt Mask Register bit 0. In all cases, setting a bit enables a function, while clearing a bit disables it.

5.4 Timer Related Sections

The High Speed I/O unit is coupled to the timers in that the HSI records the value on Timer 1 when transitions occur and the HSO causes transitions to occur based on values of either Timer 1 or Timer 2. The baud rate generator can use the T2CLK pin as input to its counter. a complete listing of the functions of IOS1, IOC0, and IOC1 are in Section 11.

6.0 HIGH SPEED INPUTS

The High Speed Input Unit (HSI), can be used to record the time at which an event occurs with respect to Timer 1. There are 4 lines (HSI.0 through HSI.3) which can be used in this mode and up to a total of 8 events can be recorded. HSI.2 and HSI.3 are bidirectional pins which can also be used as HSO.4 and HSO.5. The I/O Control Registers (IOC0 and IOC1) are used to determine the functions of these pins. A block diagram of the HSI unit is shown in Figure 24.

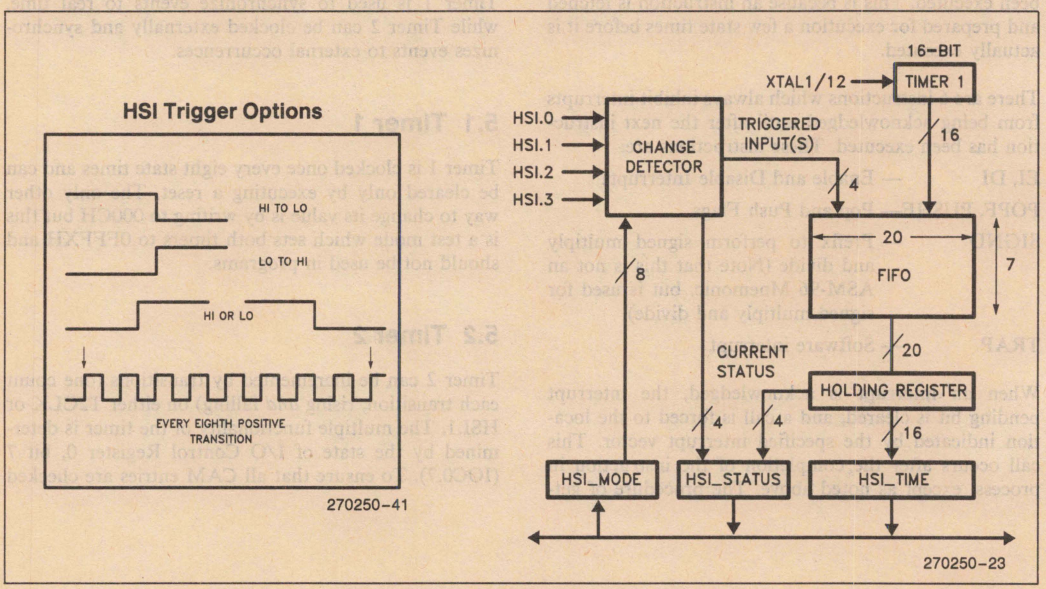


Figure 24. High Speed Input Unit

6.1 HSI Modes

There are 4 possible modes of operation for each of the HSI pins. The HSI mode register is used to control which pins will look for what type of events. The 8-bit register is set up as shown in Figure 25.

High and low levels each need to be held for at least 1 state time to ensure proper operation. The maximum input speed is 1 event every 8 state times except when the 8 transition mode is used, in which case it is 1 transition per state time.

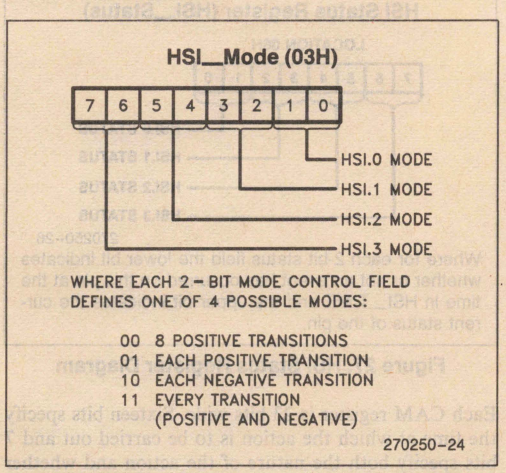


Figure 25. HSI Mode Register Diagram

The HSI lines can be individually enabled and disabled using bits in IOC0, at location 0015H. Figure 26 shows the bit locations which control the HSI pins. If the pin is disabled, transitions will not be entered in the FIFO.

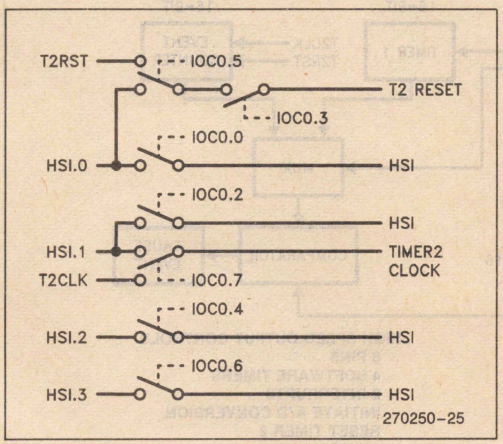


Figure 26. IOC0 Control of HSI Pin Functions

6.2 HSI FIFO

When an HSI event occurs, a 7×20 FIFO stores the 16 bits of Timer 1 and the 4 bits indicating the state of the 4 HSI pins at the time the status is read. It can take up to 8 state times for this information to reach the holding register. For this reason, 8 state times must be allowed between consecutive reads of HSI_TIME. When the FIFO is full, one additional event, for a total of 8 events, can be stored by considering the holding register part of the FIFO. If the FIFO and holding register are full, any additional events will not be recorded.

6.3 HSI Interrupts

Interrupts can be generated by the HSI unit in three ways; two FIFO related interrupts and 0 to 1 transitions on the HSI.0 pin. The HSI.0 pin can generate interrupts even if it is not enabled to the HSI FIFO. Interrupts generated by this pin cause a vector through location 2008H. The FIFO related interrupts are controlled by bit 7 of I/O Control Register 1, (IOC1.7). If the bit is a 0, then an interrupt will be generated every time a value is loaded into the holding register. If it is a 1, an interrupt will only be generated when the FIFO, (independent of the holding register), has six entries in it. Since all interrupts are rising edge triggered, if IOC1.7 = 1, the processor will not be re-interrupted until the FIFO first contains 5 or less records, then contains six or more.

6.4 HSI Status

Bits 6 and 7 of the I/O Status register 1 (IOS1) indicate the status of the HSI FIFO. If bit 6 is a 1, the FIFO contains at least six entries. If bit 7 is a 1, the FIFO contains at least 1 entry and the HSI holding register has data available to be read. The FIFO may be read after verifying that it contains valid data. Caution must be used when reading or testing bits in ISO1, as this action clears bits 0–5, including the software and hardware timer overflow flags. It is best to store the byte and then test the stored value. See Section 11.

Reading the HSI is done in two steps. First, the HSI Status register is read to obtain the current state of the HSI pins and which pins had changed at the recorded time. The format of the HSI_STATUS Register is shown in Figure 27. Second, the HSI Time register is read. Reading the Time register unloads one level of the FIFO, so if the Time register is read before the Status register, the event information in the Status register will be lost. The HSI Status register is at location 06H and the HSI Time registers are in locations 04H and 05H.

If the HSI_TIME register is read without the holding register being loaded, the returned value will be indeterminate. Under the same conditions, the four bits in HSI_STATUS indicating which events have occurred will also be indeterminate. The four HSI_STATUS bits which indicate the current state of the pins will always return the correct value.

It should be noted that many of the Status register conditions are changed by a reset, see Section 13. A complete listing of the functions of IOS0, IOS1, and IOC1 can be found in Section 11.

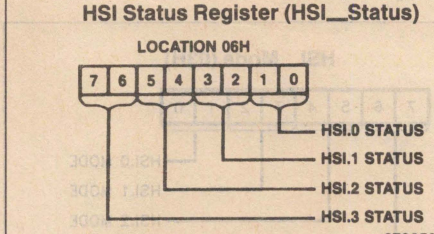
7.0 HIGH SPEED OUTPUTS

The High Speed Output unit, (HSO), is used to trigger events at specific times with minimal CPU overhead. These events include: starting an A to D conversion, resetting Timer 2, setting 4 software flags, and switching 6 output lines (HSO.0 through HSO.5). Up to eight events can be pending at one time and interrupts can be generated whenever any of these events are triggered. HSO.4 and HSO.5 are bidirectional pins which can also be used as HSI.2 and HSI.3 respectively. Bits 4 and 6 of I/O Control Register 1, (IOC1.4, IOC1.6), enable HSO.4 and HSO.5 as outputs.

The HSO unit can generate two types of interrupts. The HSO execution interrupt (vector = (2006H)) is generated (if enabled) for HSO commands which operate one or more of the six output pins. The other HSO interrupt is the software timer interrupt (vector = (200BH)) which is generated (if enabled) by any other HSO command, (e.g. triggering the A/D, resetting Timer 2 or generating a software time delay).

7.1 HSO CAM

A block diagram of the HSO unit is shown in Figure 28. The Content Addressable Memory (CAM) file is the center of control. One CAM register is compared with the timer values every state time, taking 8 state times to compare all CAM registers with the timers. This defines the time resolution of the HSO to be 8 state times (2.0 microseconds at an oscillator frequency of 12 MHz).



Where for each 2-bit status field the lower bit indicates whether or not an event has occurred on this pin at the time in HSI_TIME and the upper bit indicates the current status of the pin.

Figure 27. HSI Status Register Diagram

Each CAM register is 23 bits wide. Sixteen bits specify the time at which the action is to be carried out and 7 bits specify both the nature of the action and whether Timer 1 or Timer 2 is the reference. The format of the command to the HSO unit is shown in Figure 29. Note that bit 5 is ignored for command channels 8 through 0FH.

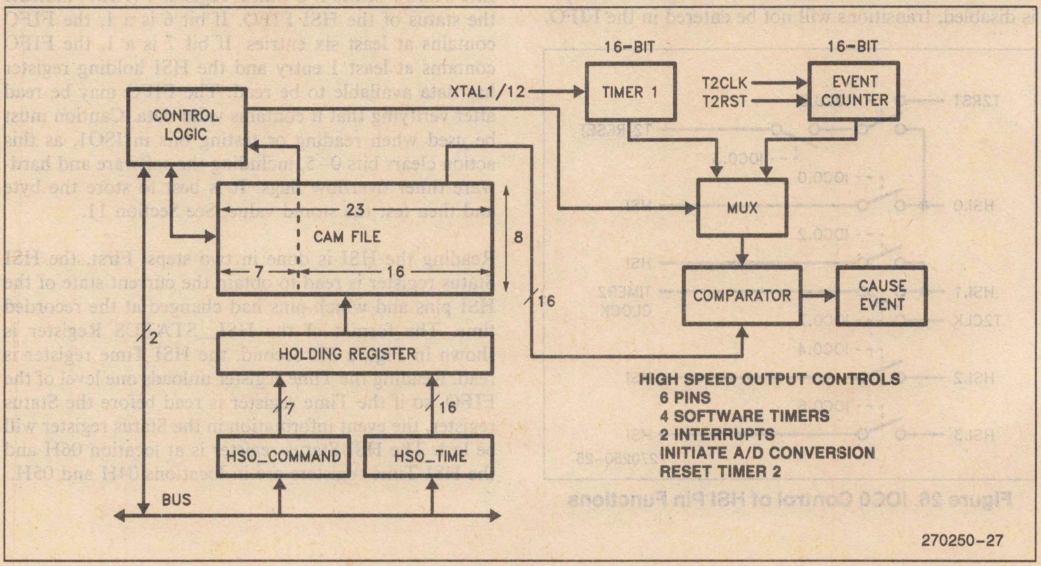


Figure 28. High Speed Output Unit

To enter a command into the CAM file, write the 7-bit "Command Tag" into location 0006H followed by the time at which the action is to be carried out into word address 0004H. The typical code would be:

```
LDB HSO_COMMAND,#what_to_do
ADD HSO_TIME,TIMER1,#when_to_do_it
```

Writing the time value loads the HSO Holding Register with both the time and the last written command tag. The command does not actually enter the CAM file until an empty CAM register becomes available.

Commands in the holding register will not execute even if their time tag is reached. Commands must be in the CAM for this to occur. Commands in the holding register can also be overwritten. Since it can take up to 8 state times for a command to move from the holding register to the CAM, 8 states must be allowed between successive writes to the CAM.

To provide proper synchronization, the minimum time that should be loaded to Timer 1 is Timer 1 + 2. Smaller values may cause the Timer match to occur 65,636 counts later than expected. A similar restriction applies if Timer 2 is used.

Care must be taken when writing the command tag for the HSO. If an interrupt occurs during the time between writing the command tag and loading the time value, and the interrupt service routine writes to the HSO time register, the command tag used in the interrupt routine will be written to the CAM at both the time specified by the interrupt routine and the time specified by the main program. The command tag from the main program will not be executed. One way of avoiding this problem would be to disable interrupts when writing commands and times to the HSO unit. See also Section 4.5.

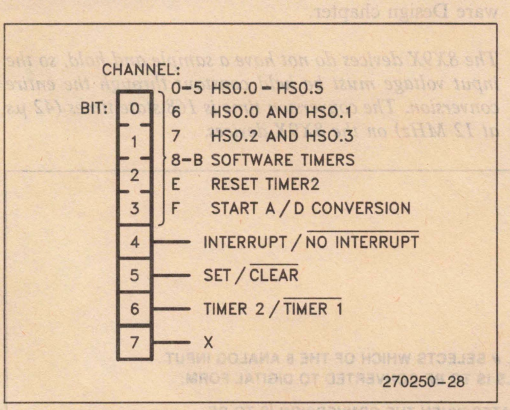


Figure 29. HSO Command Tag Format

7.2 HSO Status

Before writing to the HSO, it is desirable to ensure that the Holding Register is empty. If it is not, writing to the HSO will overwrite the value in the Holding Register. I/O Status Register 0 (IOS0) bits 6 and 7 indicate the status of the HSO unit. This register is described in Section 11. If ISO0.6 equals 0, the holding register is empty and at least one CAM register is empty. If IOS0.7 equals 0, the holding register is empty.

The programmer should carefully decide which of these two flags is the best to use for each application.

7.3 Clearing the HSO

All 8 CAM locations of the HSO are compared before any action is taken. This allows a pending external event to be cancelled by simply writing the opposite event to the CAM. However, once an entry is placed in the CAM, it cannot be removed until either the specified timer matches the written value or the chip is reset. If, as an example, a command has been issued to set HSO.1 when TIMER 1 = 1234, then entering a second command which clears HSO.1 when TIMER 1 = 1234 will result in no operation on HSO.1. Both commands will remain in the CAM until TIMER 1 = 1234.

Internal events are not synchronized to Timer 1, and therefore cannot be cleared. This includes events on HSO channels 8 through F and all interrupts. Since interrupts are not synchronized it is possible to have multiple interrupts at the same time value.

7.4 Using Timer 2 with the HSO

Timer 1 is incremented only once every 8 state-times. When it is being used as the reference timer for an HSO action, the comparator has a chance to look at all 8 CAM registers before Timer 1 changes its value. Following the same reasoning, Timer 2 has been synchronized to allow it to change at a maximum rate of once per 8 state-times. Timer 2 increments on both edges of the input signal.

When using Timer 2 as the HSO reference, caution must be taken that Timer 2 is not reset prior to the highest value for a Timer 2 match in the CAM. This is because the HSO CAM will hold an event pending until a time match occurs, if that match is to a time value on Timer 2 which is never reached, the event will remain pending in the CAM until the part is reset.

Additional caution must be used when Timer 2 is being reset using the HSO unit, since resetting Timer 2 using the HSO is an internal event and can therefore happen at any time within the eight-state-time window. For

this reason, any events scheduled to occur at the same time as a Timer 2 reset should be logged into the CAM with a Timer 2 value of zero. When using this method to make a programmable modulo counter, the count will stay at the maximum Timer 2 value only until the Reset T2 command is recognized. The count will stay at zero for the transition which would have changed the count from "N" to zero, and then changed to a one on the next transition.

7.5 Software Timers

The HSO can be programmed to generate interrupts at preset times. Up to four such "Software Timers" can be in operation at a time. As each preprogrammed time is reached, the HSO unit sets a Software Timer Flag. If the interrupt bit in the command tag was set then a Software Timer Interrupt will also be generated. The interrupt service routine can then examine I/O Status register 1 (IOS1) to determine which software timer expired and caused the interrupt. When the HSO resets Timer 2 or starts an A to D conversion, it can also be programmed to generate a software timer interrupt but there is no flag to indicate that this has occurred.

If more than one software timer interrupt occurs in the same time frame it is possible that multiple software timer interrupts will be generated.

Each read or test of any bit in IOS1 will clear bits 0 through 5. Be certain to save the byte before testing it unless you are only concerned with 1 bit. See also Section 11.5.

A complete listing of the functions of IOS0, IOS1, and IOC1 can be found in Section 11. The Timers are described in Section 5 and the HSI is described in Section 6.

8.0 ANALOG INTERFACE

The 8096H can easily interface to analog signals using its Analog to Digital Converter and its Pulse-Width-Modulated (PWM) output and HSO Unit. Analog inputs are accepted by the 8-input, 10-bit A to D converter. The PWM and HSO units provide digital signals which can be filtered for use as analog outputs.

8.1 Analog Inputs

A to D conversion is performed on one of the 8 inputs at a time using successive approximation with a result equal to the ratio of the input voltage divided by the analog supply voltage. If the ratio is 1.00, then the result will be all ones. The A/D converter is available on selected members of the MCS-96 family. See Section 14 for the device selection matrix.

Each conversion on the 8096BH requires 88 state-times (22 μs at 12 MHz) independent of the accuracy desired or value of input voltage. The input voltage must be in the range of 0 to VREF, the analog reference and supply voltage. For proper operation, VREF (the reference voltage and analog power supply) must be held nominally at 5V. The A/D result is calculated from the formula:

$$1023 \times (\text{input voltage-ANGND})/(\text{VREF-ANGND})$$

It can be seen from this formula that changes in VREF or ANGND effect the output of the converter. This can be advantageous if a ratiometric sensor is used since these sensors have an output that can be measured as a proportion of VREF.

ANGND must be tied to VSS (digital ground) in order for the 8096BH to operate properly.

A sample and hold is provided on the A/D converter of the 8X97BH and 8X95BH. The sampling window is open for 4 state times which are included in the 88 state-time conversion period. The exact timings of the A/D converter can be found in Section 3 of the Hardware Design chapter.

The 8X9X devices do not have a sample and hold, so the input voltage must be held constant through the entire conversion. The conversion time is 168 state times (42 μs at 12 MHz) on the 8X9X devices.

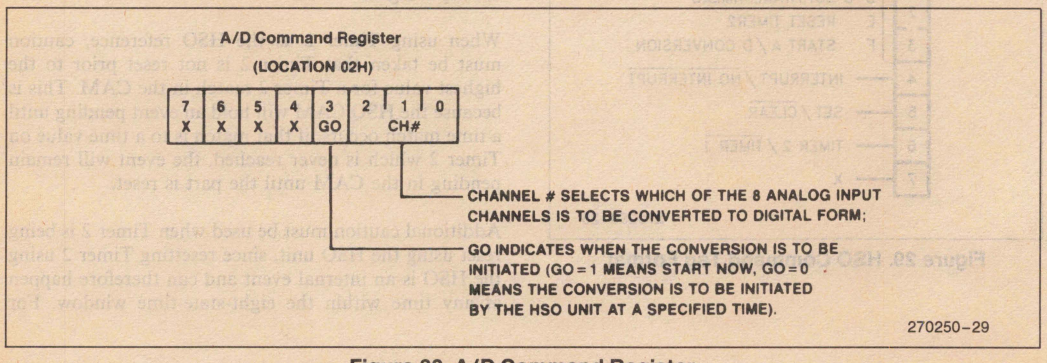


Figure 30. A/D Command Register

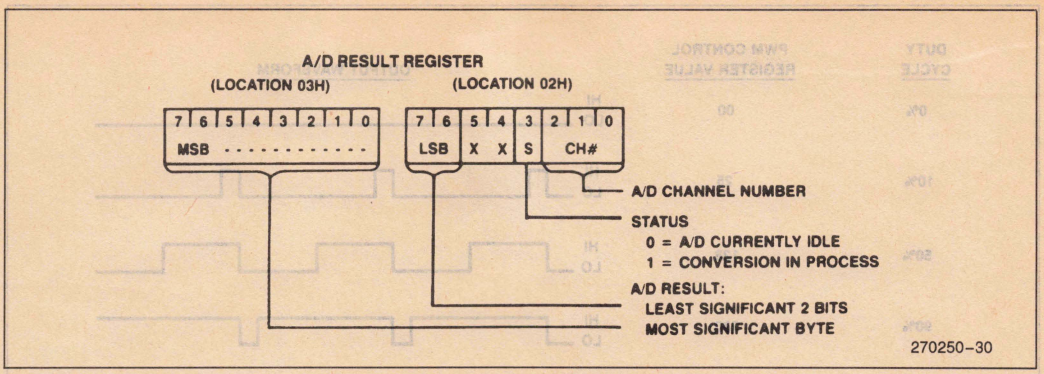


Figure 31. A/D Result Register

8.2 A/D Commands

Analog signals can be sampled by any one of the 8 analog input pins (ACH0 through ACH7) which are shared with Port 0. ACH7 can also be used as an external interrupt if IOC1.1 is set (see Sections 4 and 11). The A/D Command Register, at location 02H, selects which channel is to be converted and whether the conversion should start immediately or when the HSO (Channel #0FH) triggers it. The A/D command register must be written to for each conversion, even if the HSO is used as the trigger. A to D commands are formatted as shown in Figure 30.

The command register is double buffered so it is possible to write a command to start a conversion triggered by the HSO while one is still in progress. Care must be taken when this is done since if a new conversion is started while one is already in progress, the conversion in progress is cancelled and the new one is started. When a conversion is started, the result register is cleared. For this reason the result register must be read before a new conversion is started or data will be lost.

8.3 A/D Results

Results of the analog conversions are read from the A/D Result Register at locations 02H and 03H. Although these addresses are on a word boundary, they must be read as individual bytes. Information in the A/D Result register is formatted as shown in Figure 31. Note that the status bit may not be set until 8 state times after the go command, so it is necessary to wait 8 state times before testing it. Information on using the HSO is in Section 7.

8.4 Pulse Width Modulation Output (D/A)

Digital to analog conversion can be done with the Pulse Width Modulation output; a block diagram of the circuit is shown in Figure 32. The 8-bit counter is incremented every state time. When it equals 0, the PWM output is set to a one. When the counter matches the value in the PWM register, the output is switched low. When the counter overflows, the output is once again switched high. A typical output waveform is shown in Figure 33. Note that when the PWM register equals 00, the output is always low.

The output waveform is a variable duty cycle pulse which repeats every 256 state times (64 μs at 12 MHz). Changes in the duty cycle are made by writing to the PWM register at location 17H. There are several types

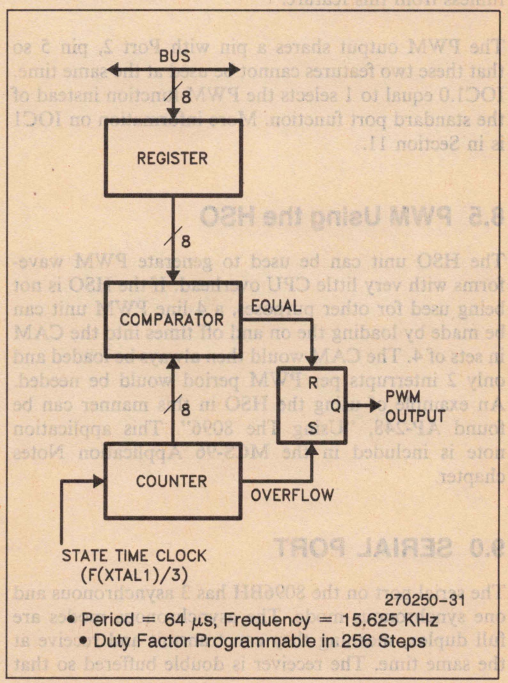


Figure 32. Pulse Width Modulated (D/A) Output

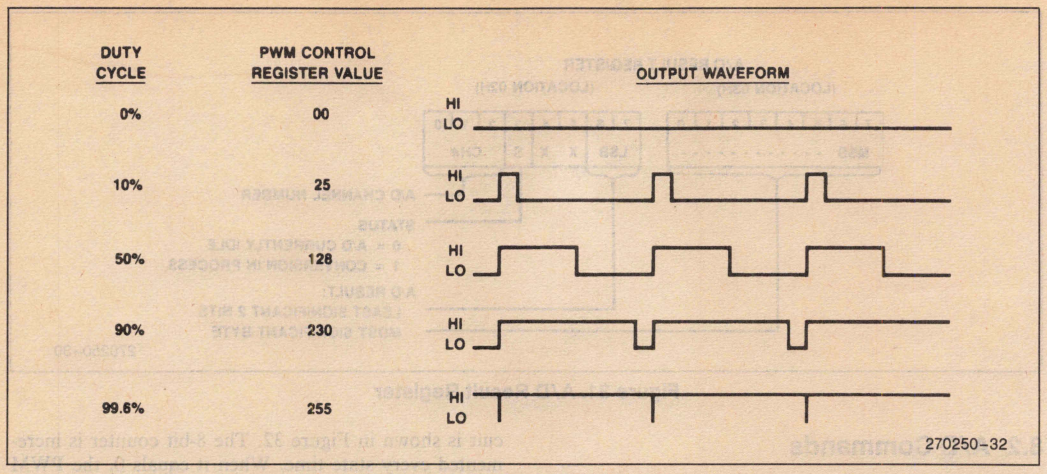


Figure 33. Typical PWM Outputs

of motors which require a PWM waveform for most efficient operation. Additionally, if this waveform is integrated it will produce a DC level which can be changed in 256 steps by varying the duty cycle.

Details about the hardware required for smooth, accurate D/A conversion can be found in Section 4 of the Hardware Design chapter. Typically, some form of buffer and integrator are needed to obtain the most usefulness from this feature.

The PWM output shares a pin with Port 2, pin 5 so that these two features cannot be used at the same time. IOC1.0 equal to 1 selects the PWM function instead of the standard port function. More information on IOC1 is in Section 11.

8.5 PWM Using the HSO

The HSO unit can be used to generate PWM waveforms with very little CPU overhead. If the HSO is not being used for other purposes, a 4 line PWM unit can be made by loading the on and off times into the CAM in sets of 4. The CAM would then always be loaded and only 2 interrupts per PWM period would be needed. An example of using the HSO in this manner can be found AP-248, "Using The 8096". This application note is included in the MCS-96 Application Notes chapter.

9.0 SERIAL PORT

The serial port on the 8096BH has 3 asynchronous and one synchronous mode. The asynchronous modes are full duplex, meaning they can transmit and receive at the same time. The receiver is double buffered so that the reception of a second byte can begin before the first byte has been read. The port is functionally compatible with the serial port on the MCS-51 family of microcontrollers, although the software used to control the ports is different.

Control of the serial port is handled through the Serial Port Control/Status Register at location 11H. Figure 37 shows the layout of this register. The details of using it to control the serial port will be discussed in Section 9.2.

Data to and from the serial port is transferred through SBUF (rx) and SBUF (tx), both located at 07H. Although these registers share the same address, they are physically separate, with SBUF (rx) containing the data received by the serial port and SBUF (tx) used to hold data ready for transmission. The program cannot write to SBUF (rx) or read from SBUF (tx).

The baud rate at which the serial port operates is controlled by an independent baud rate generator. The inputs to this generator can be either the XTAL1 or the T2CLK pin. Details on setting up the baud rate are given in Section 9.3.

9.1 Serial Port Modes

MODE 0

Mode 0 is a synchronous mode which is commonly used for shift register based I/O expansion. In this mode the TXD pin outputs a set of 8 pulses while the RXD pin either transmits or receives data. Data is transferred 8 bits at a time with the LSB first. A diagram of the relative timing of these signals is shown in Figure 34.

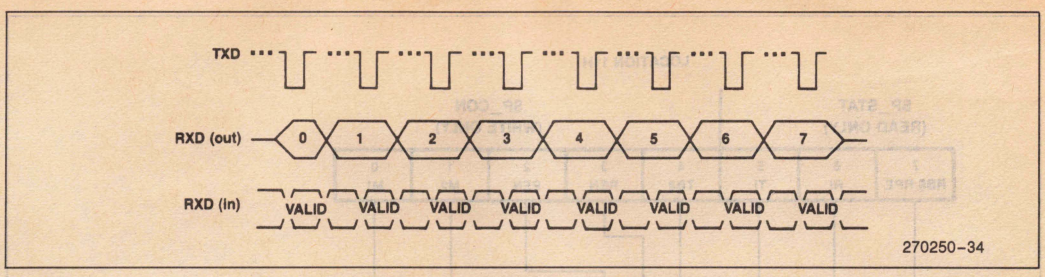


Figure 34. Serial Port Mode 0 Timing

Although it is not possible to transmit and receive at the same time using this mode, two external gates and a port pin can be used to time-multiplex the two functions. An example of multiplexing transmit and receive is discussed in Section 6.1 of the Hardware Design chapter.

MODE 1

Mode 1 is the standard asynchronous communications mode. The data frame used in this mode is shown in Figure 35. It consists of 10 bits; a start bit (0), 8 data bits (LSB first), and a stop bit (1). If parity is enabled, (the PEN bit is set to a 1), an even parity bit is sent instead of the 8th data bit and parity is checked on reception.

MODE 2

Mode 2 is the asynchronous 9th bit recognition mode. This mode is commonly used with Mode 3 for multiprocessor communications. Figure 36 shows the data frame used in this mode. It consists of a start bit (0), 9 data bits (LSB first), and a stop bit (1). When transmitting, the 9th bit can be set to a one by setting the TB8 bit in the control register before writing to SBUF (tx). The TB8 bit is cleared on every transmission, so it must be set prior to writing to SBUF (tx) each time it is desired. During reception, the serial port interrupt and the Receive Interrupt (RI) bit will not be set unless the 9th bit being received is set. This provides an easy way to have selective reception on a data link. Parity cannot be enabled in this mode.

MODE 3

Mode 3 is the asynchronous 9th bit mode. The data frame for this mode is identical to that of Mode 2. The transmission differences between Mode 3 and Mode 2 are that parity can be enabled (PEN = 1) and cause the 9th data bit to take the even parity value. The TB8 bit can still be used if parity is not enabled (PEN = 0). When in Mode 3, a reception always causes an interrupt, regardless of the state of the 9th bit. The 9th bit is stored if PEN = 0 and can be read in bit RB8. If PEN = 1 then RB8 becomes the Receive Parity Error (RPE) flag.

9.2 Controlling the Serial Port

Control of the serial port is done through the Serial Port Control (SP_CON) and Serial Port Status (SP_STAT) registers shown in Figure 37. Writing to location 11H accesses SP_CON while reading it access SP_STAT. Note that reads of SP_STAT will return indeterminate data in the lower 5 bits and writing

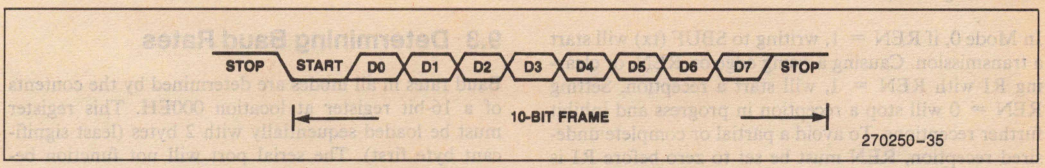


Figure 35. Serial Port Frame—Mode 1

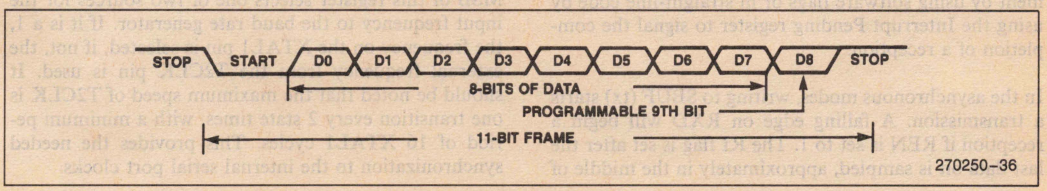
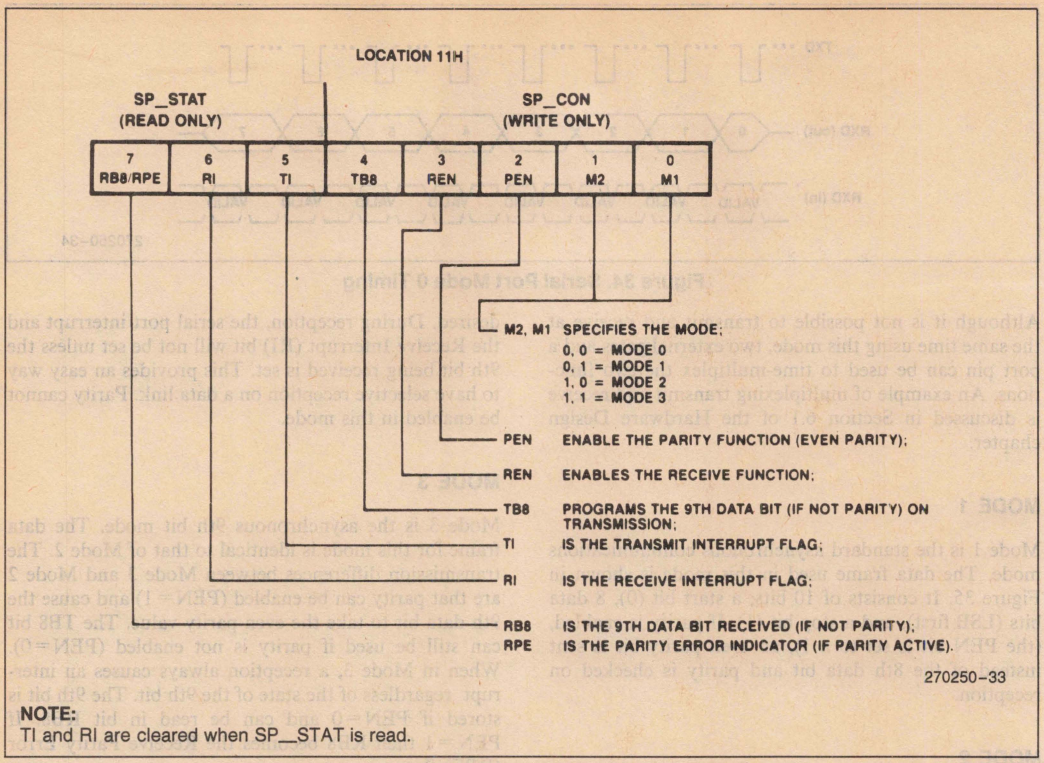


Figure 36. Serial Port Frame Modes 2 and 3



NOTE:
TI and RI are cleared when SP_STAT is read.

Figure 37. Serial Port Control/Status Register

to the upper 3 bits of SP_CON has no effect on chip functionality. The TB8 bit is cleared after each transmission and both TI and RI are cleared whenever SP_STAT (not SP_CON) is accessed. Whenever the TXD pin is used for the serial port it must be enabled by setting IOC1.5 to a 1. IOC1 is discussed further in Section 11.3. Information on the hardware connections and timing of the serial port is in Section 6 of the Hardware Design chapter.

In Mode 0, if REN = 1, writing to SBUF (tx) will start a transmission. Causing a rising edge on REN, or clearing RI with REN = 1, will start a reception. Setting REN = 0 will stop a reception in progress and inhibit further receptions. To avoid a partial or complete undesired reception, REN must be set to zero before RI is cleared. This can be handled in an interrupt environment by using software flags or in straight-line code by using the Interrupt Pending register to signal the completion of a reception.

In the asynchronous modes, writing to SBUF (tx) starts a transmission. A falling edge on RXD will begin a reception if REN is set to 1. The RI flag is set after the last data bit is sampled, approximately in the middle of the bit time. The TI flag is set after the last data bit (either 8th or 9th) is sent, also in the middle of the bit time. New data placed in SBUF (tx) is held and will not be transmitted until the end of the stop bit has been sent. The flags clear when SP_STAT is read, but do not have to be clear for the port to receive or transmit. The serial port interrupt bit is set as a logical OR of the RI and TI bits.

9.3 Determining Baud Rates

Baud rates in all modes are determined by the contents of a 16-bit register at location 000EH. This register must be loaded sequentially with 2 bytes (least significant byte first). The serial port will not function between the loading of the first and second bytes. The MSB of this register selects one of two sources for the input frequency to the baud rate generator. If it is a 1, the frequency on the XTAL1 pin is selected, if not, the external frequency from the T2CLK pin is used. It should be noted that the maximum speed of T2CLK is one transition every 2 state times, with a minimum period of 16 XTAL1 cycles. This provides the needed synchronization to the internal serial port clocks.

The unsigned integer represented by the lower 15 bits of the baud rate register defines a number B, where B has a maximum value of 32767. The baud rate for the four serial modes using either XTAL1 or T2CLK as the clock source is given by:

Using XTAL1:

$$\text{Mode 0: } \frac{\text{Baud}}{\text{Rate}} = \frac{\text{XTAL1 frequency}}{4 * (B + 1)}; \quad B \neq 0$$

$$\text{Others: } \frac{\text{Baud}}{\text{Rate}} = \frac{\text{XTAL1 frequency}}{64 * (B + 1)}$$

Using T2CLK:

$$\text{Mode 0: } \frac{\text{Baud}}{\text{Rate}} = \frac{\text{T2CLK frequency}}{B}; \quad B \neq 0$$

$$\text{Others: } \frac{\text{Baud}}{\text{Rate}} = \frac{\text{T2CLK frequency}}{16 * B}; \quad B \neq 0$$

Note that B cannot equal 0, except when using XTAL1 in other than mode 0.

Common baud rate values, using XTAL1 at 12 MHz, are shown below.

Baud Rate	Baud Register Value	
	Mode 0	Others
9600	8137H	8013H
4800	8270H	8026H
2400	84E1H	804DH
1200	89C3H	809BH
300	A70FH	8270H

The maximum baud rates are 1.5 Mbaud synchronous and 187.5 Kbaud asynchronous with 12 MHz on XTAL1.

9.4 Multiprocessor Communications

Mode 2 and 3 are provided for multiprocessor communications. In Mode 2 if the received 9th data bit is not 1, the serial port interrupt is not activated. The way to use this feature in multiprocessor systems is described below.

When the master processor wants to transmit a block of data to one of several slaves, it first sends out an address frame which identifies the target slave. An address frame will differ from a data frame in that the 9th data bit is 1 in an address frame and 0 in a data frame. No slave in Mode 2 will be interrupted by a data frame.

An address frame, however, will interrupt all slaves so that each slave can examine the received byte and see if it is being addressed. The addressed slave switches to Mode 3 to receive the coming data frames, while the slaves that were not addressed stay in Mode 2 and go on about their business.

10.0 I/O PORTS

There are five 8-bit I/O ports on the 8096. Some of these ports are input only, some are output only, some are bidirectional and some have alternate functions. In addition to these ports, the HSI/O unit can be used to provide extra I/O lines if the timer related features of these lines are not needed.

Input ports connect to the internal bus through an input buffer. Output ports connect through an output buffer to an internal register that hold the bits to be output. Bidirectional ports consist of an internal register, an input buffer, and an output buffer.

Port 0 is an input port which is also used as the analog input for the A to D converter. Port 1 is a quasi-bidirectional port. Port 2 contains three types of port lines: quasi-bidirectional, input and output. The input and output lines are shared with other functions in the 8096BH as shown in Table 4. Ports 3 and 4 are open-drain bidirectional ports which share their pins with the address/data bus.

Table 4. Port 2 Alternate Functions

Port	Function	Alternate Function	Controlled by
P2.0	Output	TXD (Serial Port Transmit)	IOC1.5
P2.1	Input	RXD (Serial Port Receive)	N/A
P2.2	Input	EXTINT (External Interrupt)	IOC1.1
P2.3	Input	T2CLK (Timer 2 Input)	IOC0.7
P2.4	Input	T2RST (Timer 2 Reset)	IOC0.5
P2.5	Output	PWM (Pulse-Width Modulation)	IOC1.0
P2.6	Quasi-Bidirectional		
P2.7	Quasi-Bidirectional		

Section 2 of the Hardware Design chapter contains additional information on the timing, drive capabilities, and input impedances of I/O pins.

10.1 Input Ports

Input ports and pins can only be read. There are no output drivers on these pins. The input leakage of these pins is in the microamp range. The specific values can be found in the data sheet for the device being considered.

In addition to acting as a digital input, each line of Port 0 can be selected to be the input of the A to D converter as discussed in Section 8. The pins on Port 0 are tested to have D.C. leakage of 3 microamps or less, as specified in the data sheet for the device being considered. The capacitance on these pins is approximately 5 pF and will instantaneously increase by around 5 pF when the pin is being sampled by the A to D converter.

The 8096BH samples the input to the A/D for 4 state times at the beginning of the conversion. *The 8X9X devices sample the A/D pin 10 times during a conversion.* Details on the A to D converter can be found in Section 8 of this chapter and in Section 3 of the Hardware Design chapter.

10.2 Quasi-Bidirectional Ports

Port 1, Port 2.6 and Port 2.7 are quasi-bidirectional ports. "Quasi-bidirectional" means that the port pin has a weak internal pullup that is always active and an internal pulldown which can be on to output a 0, or off to output a 1. If the internal pulldown is left off (by writing a 1 to the pin), the pin's logic level can be controlled by an external pulldown. If the external pulldown is on, it will input a 0 to the 8096BH, if it is off, a 1 will be input. From the user's point of view, the main difference between a quasi-bidirectional port and a standard input port is that the quasi-bidirectional port will source current if externally pulled low. It will also pull itself high if left unconnected.

In parallel with the weak internal pullup is a much stronger internal pullup that is activated for one state time when the pin is internally driven from 0 to 1. This is done to speed up the 0-to-1 transition time. When this pullup is on the pin can typically source 30 milliamps to V_{SS}.

When the processor writes to the pins of a quasi-bidirectional port it actually writes into a register which in turn drives the port pin. When the processor reads these ports, it senses the status of the pin directly. If a port pin is to be used as an input then the software should write a one to that pin, this will cause the low-impedance pull-down device to turn off and leave the pin pulled up with a relatively high impedance pullup device which can be easily driven down by the device driving the input.

If some pins of a port are to be used as inputs and some are to be used as outputs the programmer should be careful when writing to the port. Consider using P1.0 as an input and then trying to toggle P1.1 as an output:

```
ORB    IOPORT1,#00000001B ;Set P1.0
                           for input
XORB   IOPORT1,#00000010B ;Complement
                           P1.1
```

The first instruction will work as expected but two problems can occur when the second instruction executes. The first is that even though P1.1 is being driven high by the 8096 it is possible that it is being held low externally. This typically happens when the port pin is used to drive the base of an NPN transistor which in turn drives whatever there is in the outside world which needs to be toggled. The base of the transistor will clamp the port pin to the transistor's Vbe above ground, typically 0.7V. The 8096 will input this value as a zero even if a one has been written to the port pin. When this happens the XORB instruction will always write a one to the port pin and it will not toggle. The second problem, which is related to the first one, is that if P1.0 happens to be driven to a zero when Port 1 is read by the XORB instruction then the XORB will write a zero to P1.0 and it will no longer be useable as an input. The first problem can best be solved by the external driver design. A series resistor between the port pin and the base of the transistor often works. The second problem can be solved in the software fairly easily:

```
LDB    AL,IOPORT1
XORB   AL,#010B
ORB    AL,#001B
STB    AL,IOPORT1
```

A software solution to both problems is to keep a byte in RAM as an image of the data to be output to the port; any time the software wants to modify the data on the port it can then modify the image byte and then copy it to the port.

When using the quasi-bidirectional ports as inputs tied to switches, series resistors may be needed if the ports will be written to internally after the part is initialized. The amount of current sourced to ground by each pin is typically 20 mA or more. Therefore, if all 8 pins are tied to ground 160 mA will be sourced. This is equivalent to instantaneously doubling the power used by the chip and may cause noise in some applications.

This potential problem can be solved easily in software by never writing a zero to a pin being used as an input.

10.3 Output Ports

Output pins include the bus control lines, the HSO lines, and some of Port 2. These pins can only be used as outputs as there are no input buffers connected to them. It is not possible to use immediate logical instructions such as XOR PORT2, #00111B to toggle these pins. The output currents on these ports is higher than that of the quasi-bidirectional ports.

10.4 Ports 3 and 4/AD0–15

These pins have two functions. They are either bidirectional ports with open-drain outputs or System Bus pins which the memory controller uses when it is accesing off-chip memory. If the $\overline{EA}$ line is low, the pins always act as the System Bus. Otherwise they act as bus pins only during a memory access. If these pins are being used as ports and bus pins, ones must be written to them prior to bus operations.

When acting as the system bus the pins have strong drivers to both V_{CC} and V_{SS}. These drivers are used whenever data is being output on the system bus and are not used when data is being output by Ports 3 and 4. Only the pins and input buffers are shared between the bus and the ports. The ports use different output buffers which are configured as open-drain, and require pullup resistors. (open-drain is the MOS version of open-collector.) The port pins and their system bus functions are shown in Table 5.

Table 5. P3,4/AD0–15 Pins

Port Pin	System Bus Function
P3.0	AD0
P3.1	AD1
P3.2	AD2
P3.3	AD3
P3.4	AD4
P3.5	AD5
P3.6	AD6
P3.7	AD7
P4.0	AD8
P4.1	AD9
P4.2	AD10
P4.3	AD11
P4.4	AD12
P4.5	AD13
P4.6	AD14
P4.7	AD15

11.0 STATUS AND CONTROL REGISTERS

There are two I/O Control registers, IOC0 and IOC1. IOC0 controls Timer 2 and the HSI lines. IOC1 controls some pin functions, interrupt sources and 2 HSO pins.

Whenever input lines are switched between two sources, or enabled, it is possible to generate transitions on these lines. This could cause problems with respect to edge sensitive lines such as the HSI lines, Interrupt line, and Timer 2 control lines.

11.1 I/O Control Register 0 (IOC0)

IOC0 is located at 0015H. The four HSI lines can be enabled or disabled to the HSI unit by setting or clearing bits in IOC0. Timer 2 functions including clock and reset sources are also determined by IOC0. The control bit locations are shown in Figure 38.

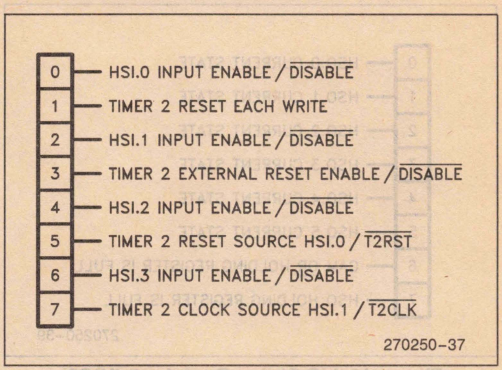


Figure 38. I/O Control Register 0 (IOC0)

11.2 I/O Control Register 1 (IOC1)

IOC1 is used to select some pin functions and enable or disable some interrupt sources. Its location is 0016H. Port pin P2.5 can be selected to be the PWM output instead of a standard output. The external interrupt source can be selected to be either EXTINT (same pin as P2.2) or Analog Channel 7 (ACH7, same pin as P0.7). Timer 1 and Timer 2 overflow interrupts can be individually enabled or disabled. The HSI interrupt can be selected to activate either when there is 1 FIFO entry or 7. Port pin P2.0 can be selected to be the TXD output. HSO.4 and HSO.5 can be enabled or disabled to the HSO unit. More information on interrupts is available in Section 4. The positions of the IOC1 control bits are shown in Figure 39.

11.3 I/O Status Register 0 (IOS0)

There are two I/O Status registers, IOS0 and IOS1. IOS0, located at 0015H, holds the current status of the HSO lines and CAM. The status bits of IOS0 are shown in Figure 40.

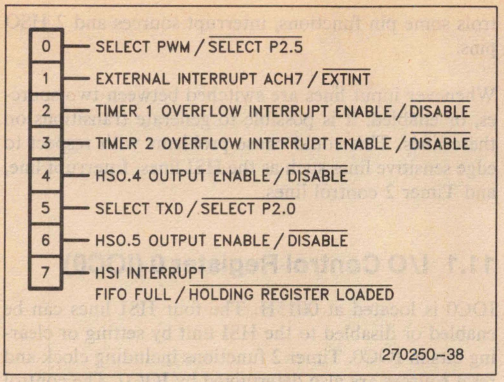


Figure 39. I/O Control Register 1 (IOC1)

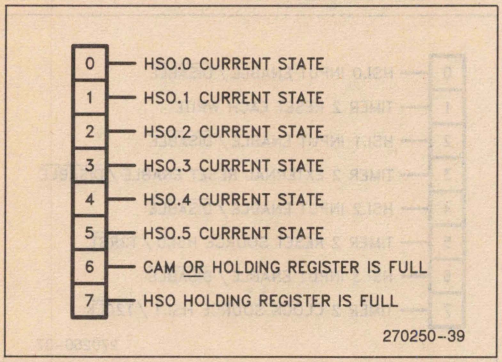


Figure 40. I/O Status Register 0 (IOS0)

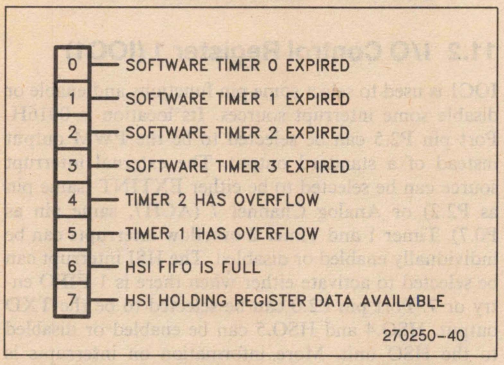


Figure 41. HSIO Status Register 1 (IOS1)

11.4 I/O Status Register 1 (IOS1)

IOS1 is located at 016H. It contains status bits for the timers and the HSI/O. The positions of these bits are shown in Figure 41.

Whenever the processor reads this register all of the time-related flags (bits 5 through 0) are cleared. This applies not only to explicit reads such as:

```
LDB    AL,IOS1
```

but also to implicit reads such as:

```
JB     IOS1.3,somewhere_else
```

which jumps to somewhere_else if bit 3 of IOS1 is set. In most cases this situation can best be handled by having a byte in the register file which is used to maintain an image of lower five bits of the register. Any time a hardware timer interrupt or a HSO software timer interrupt occurs the byte can be updated:

```
ORB    IOS1_image,IOS1
```

leaving IOS1_image containing all the flags that were set before plus all the new flags that were read and cleared from IOS1. Any other routine which needs to sample the flags can safely check IOS1_image. Note that if these routines need to clear the flags that they have acted on, then the modification of IOS1_image must be done from inside a critical region (see Section 4.4).

12.0 WATCHDOG TIMER

The WatchDog Timer (WDT) provides a means to recover gracefully from a software upset. When the watchdog is enabled it will initiate a hardware reset unless the software clears it every 64K state times.

The WDT is implemented as an 8-bit timer with an 8-bit prescaler. The prescaler is not synchronized, so the timer will overflow between 65280 and 65535 state times after being reset. When the timer overflows it pulls down the RESET pin for at least two state times, resetting the 8096BH and any other devices tied to the RESET line. If a large capacitor is connected to the line, the pin may take a long time to go low. This will effect the length of time the pin is low and the voltage on the pin when it is finished falling. Section 1.4 of the Hardware Design chapter contains more information about reset hardware connections.

The WDT is enabled the first time it is cleared. Once it is enabled, it can only be disabled by resetting the 8096BH. The internal bit which controls the watchdog can typically maintain its state through power glitches as low as V_{SS} and as high as 7.0V for up to one millisecond.

The 8X9X devices do not have the extra glitch protection on the WDT enable bit.

Enabling and clearing the WDT is done by writing a "01EH" followed by a "0E1H" to the WDT register at location 0AH. This double write is used to help prevent accidental clearing of the timer.

12.1 Software Protection Hints

Glitches and noise on the PC board can cause software upsets, typically by changing either memory locations or the program counter. These changes can be internal to the chip or be caused by bad data returning to the chip.

There are both hardware and software solutions to noise problems, but the best solution is good design practice and a few ounces of prevention. The software can be designed so that the watchdog times out if the program does not progress properly. The watchdog will also time-out if the software error was due to ESD (Electrostatic Discharge) or other hardware related problems. This prevents the controller from having a malfunction for longer than 16 milliseconds if a 12 MHz oscillator is used.

When using the WDT to protect software it is desirable to reset it from only one place in code. This will lessen the chance that an undesired WDT reset will occur. The section of code that resets the WDT should monitor the other code sections for proper operation. This can be done by checking variables to make sure they are within reasonable values. Simply using a software timer to reset the WDT every 15 milliseconds will not provide much protection against minor problems.

It is also recommended that unused areas of code be filled with NOPs and periodic jumps to an error routine or RST (reset chip) instructions. This is particularly important in the code around lookup tables, since if lookup tables are executed undesired results will occur. Wherever space allows, each table should be surrounded by 7 NOPs (the longest 8096 instruction has 7 bytes) and a RST or jump to error routine instruction. Since RST is a one-byte instruction, the NOPs are not needed if RSTs are used instead of jumps to an error routine. This will help to ensure a speedy recovery should the processor have a glitch in the program flow. Since RST instruction has an opcode of 0FFH, pulling the data lines high with resistors will cause an RST to be executed if unimplemented memory is addressed.

12.2 Disabling The Watchdog

The watchdog should be disabled by software not initializing it. If this is not possible, such as during program development, the watchdog can be disabled by holding the $\overline{\text{RESET}}$ pin at 2.0V to 2.5V. Voltages over 2.5V on the pin could quickly damage the part. Even at 2.5V, using this technique for other than debugging purposes is not recommended, as it may effect long term reliability. It is further recommended that any part used in this way for more than several seconds, not be used in production versions of products. Section 1.6 of the Hardware Design chapter has more information on disabling the Watchdog Timer.

13.0 RESET

13.1 Reset Signal

As with all processors, the 8096BH must be reset each time the power is turned on. This is done by holding the $\overline{\text{RESET}}$ pin low for at least 2 state times after the power supply is within tolerance and the oscillator has stabilized.

On 8X9X devices the $\overline{\text{RESET}}$ pin must be held low long enough for the power supply, oscillator and back-bias generator to stabilize. Typically, the back-bias generator requires one millisecond to stabilize.

After the $\overline{\text{RESET}}$ pin is brought high, a ten state reset sequence is executed. During this time, the Chip Configuration Byte (CCB) is read from location 2018H and written to the 8096BH Chip Configuration Register (CCR). If the voltage on the $\overline{\text{EA}}$ pin selects the internal/external execution mode the CCB is read from internal ROM/EPROM. If the voltage on the $\overline{\text{EA}}$ pin selects the external execution only mode the CCB is read from external memory.

The 8096BH can be reset using a capacitor, 1-shot, or any other method capable of providing a pulse of at least 2 state times longer than required for V_{CC} and the oscillator to stabilize.

For best functionality, it is suggested that the reset pin be pulled low with an open collector device. In this way, several reset sources can be wire ORed together. Remember, the $\overline{\text{RESET}}$ pin itself can be a reset source when the RST instruction is executed or when the Watchdog Timer overflows. Details of hardware suggestions for reset can be found in Section 1.4 of the Hardware Design chapter.

13.2 Reset Status

The I/O lines and control lines of the 8096BH will be in their reset state within 2 state times after reset is low, with V_{CC} and the oscillator stabilized. Prior to that time, the status of the I/O lines is indeterminate. After the 10 state time reset sequence, the Special Function Registers will be set as follows:

Register	Reset Value
Port 1	11111111B
Port 2	110XXXX1B
Port 3	11111111B
Port 4	11111111B
PWM Control	00H
Serial Port (Transmit)	undefined
Serial Port (Receive)	undefined
Baud Rate Register	undefined
Serial Port Control	XXXX0XXXB
Serial Port Status	X00XXXXXB
A/D Command	undefined
A/D Result	undefined
Interrupt Pending	undefined
Interrupt Mask	00000000B
Timer 1	0000H
Timer 2	0000H
Watchdog Timer	0000H
HSI Mode	11111111B
HSI Status	undefined
IOS0	00000000B
IOS1	00000000B
IOC0	X0X0X0X0B
IOC1	X0X0XXX1B
HSI FIFO	empty
HSO CAM	empty
HSO lines	000000B
PSW	0000H
Stack Pointer	undefined
Program Counter	2080H

Other conditions following a reset are:

Pin	Reset Value
$\overline{RD}$	high
$\overline{WR}/\overline{WRL}$	high
ALE/$\overline{ADV}$	high
$\overline{BHE}/\overline{WRH}$	low
INST	high

It is important to note that the Stack Pointer and Interrupt Pending Register are undefined, and need to be initialized in software. The Interrupts are disabled by both the mask register and PSW.9 after a reset.

13.3 Reset Sync Mode

The $\overline{RESET}$ line can be used to start the 8096BH at an exact state time to provide for synchronization of test equipment and multiple chip systems. $\overline{RESET}$ is active low. To synchronize parts, $\overline{RESET}$ is brought high on the rising edge of XTAL1. Complete details on synchronizing parts can be found in Section 1.5 of the Hardware Design chapter.

It is very possible that parts which start in sync may not stay that way. The best example of this would be when a "jump on I/O bit" is being used to hold the processor in a loop. If the line changes during the time it is being tested, one processor may see it as a one, while the other sees it as a zero. The result is that one processor will do an extra loop, thus putting it several states out of sync with the other.

14.0 QUICK REFERENCE

14.1 Pin Description

On the 48-pin parts the following pins are not bonded out: Port1, Port0 (Analog In) bits 0–3, T2CLK (P2.3), T2RST (P2.4), P2.6, P2.7, CLKOUT, INST, NMI, BUSWIDTH ($\overline{TEST}$ on 8X9X devices).

PIN DESCRIPTIONS

Symbol	Name and Function
V_{CC}	Main supply voltage (5V).
V_{SS}	Digital circuit ground (0V).
V_{PD}	RAM standby supply voltage (5V). This voltage must be present during normal operation. In a Power Down condition (i.e. V_{CC} drops to zero), if $\overline{RESET}$ is activated before V_{CC} drops below spec and V_{PD} continues to be held within spec., the top 16 bytes in the Register File will retain their contents. $\overline{RESET}$ must be held low during the Power Down and should not be brought high until V_{CC} is within spec and the oscillator has stabilized. See Section 2.3.
V_{REF}	Reference voltage for the A/D converter (5V). V_{REF} is also the supply voltage to the analog portion of the A/D converter and the logic used to read Port 0. See Section 8.
ANGND	Reference ground for the A/D converter. Should be held at nominally the same potential as V_{SS}. See Section 8.
V_{PP} V_{BB}(8X9X)	Programming voltage for the EPROM parts. It should be +12.75V when programming and will float to 5V otherwise. It should not be above 5.5V on other than EPROM parts. This pin is V_{BB} on 8X9X parts. Systems that have this pin connected to ANGND through a capacitance (required on 8X9X parts) do not need to change.
XTAL1	Input of the oscillator inverter and of the internal clock generator. See Section 1.5.
XTAL2	Output of the oscillator inverter. See Section 1.5.
CLKOUT	Output of the internal clock generator. The frequency of CLKOUT is 1/3 the oscillator frequency. It has a 33% duty cycle. See Section 1.5
$\overline{RESET}$	Reset input to the chip. Input low for at least 2 state times to reset the chip. The subsequent low-to-high transition re-synchronizes CLKOUT and commences a 10-state-time sequence in which the PSW is cleared, a byte read from 2018H loads CCR, and a jump to location 2080H is executed. Input high for normal operation. $\overline{RESET}$ has an internal pullup. (The read from 2018H is not done on 8X9X parts). See Section 13.
BUSWIDTH $\overline{TEST}$(8X9X)	Input for buswidth selection. If CCR bit 1 is a one, this pin selects the bus width for the bus cycle in progress. If BUSWIDTH is a 1, a 16-bit bus cycle occurs. If BUSWIDTH is a 0 an 8-bit cycle occurs. If CCR bit 1 is a 0, the bus is always an 8-bit bus. This pin is the $\overline{TEST}$ pin on 8X9X parts. Systems with $\overline{TEST}$ tied to V_{CC} do not need to change. If this pin is left unconnected, it will rise to V_{CC}. See Section 2.7.
NMI	A positive transition clears the Watchdog Timer count, and causes a vector to external memory location 0000H. External memory from 00H through 0FFH is reserved for Intel development systems.
INST	Output high during an external memory read indicates the read is an instruction fetch. INST is valid throughout the bus cycle.
$\overline{EA}$	Input for memory select (External Access). $\overline{EA}$ equal to a TTL-high causes memory accesses to locations 2000H through 3FFFH to be directed to on-chip ROM/EPROM. $\overline{EA}$ equal to a TTL-low causes accesses to these locations to be directed to off-chip memory. $\overline{EA}$ = +12.5V causes execution to begin in the Programming mode on EPROM parts. $\overline{EA}$ has an internal pulldown, so it goes to 0 unless driven otherwise.
ALE/$\overline{ADV}$	Address Latch Enable or Address Valid output, as selected by CCR. Both pin options provide a latch to demultiplex the address from the address/data bus. When the pin is $\overline{ADV}$, it goes inactive high at the end of the bus cycle. $\overline{ADV}$ can be used as a chip select for external memory. ALE/$\overline{ADV}$ is activated only during external memory accesses. (The $\overline{ADV}$ function is not available on 8X9X parts). See Section 2.7.
$\overline{RD}$	Read signal output to external memory. $\overline{RD}$ is activated only during external memory reads.

PIN DESCRIPTIONS (Continued)

Symbol	Name and Function
$\overline{WR}/\overline{WRL}$	Write and Write Low output to external memory, as selected by the CCR. $\overline{WR}$ will go low for every external write, while $\overline{WRL}$ will go low only for external writes where an even byte is being written. $\overline{WR}/\overline{WRL}$ is activated only during external memory writes. (The $\overline{WRL}$ function is not available on 8X9X parts). See Section 2.7.
$\overline{BHE}/\overline{WRH}$	Bus High Enable or Write High output to external memory, as selected by the CCR. $\overline{BHE}$ = 0 selects the bank of memory that is connected to the high byte of the data bus. A0 = 0 selects the bank of memory that is connected to the low byte of the data bus. Thus accesses to a 16-bit wide memory can be to the low byte only (A0 = 0, $\overline{BHE}$ = 1), to the high byte only (A0 = 1, BHE # = 0), or both bytes (A0 = 0, $\overline{BHE}$ = 0). If the $\overline{WRH}$ function is selected, the pin will go low if the bus cycle is writing to an odd memory location. (The $\overline{WRH}$ function is not available on 8X9X parts). See Section 2.7.
READY	Ready input to lengthen external memory cycles, for interfacing to slow or dynamic memory, or for bus sharing. If the pin is high, CPU operation continues in a normal manner. If the pin is low prior to the falling edge of CLKOUT, the Memory Controller goes into a wait mode until the next positive transition in CLKOUT occurs with READY high. The bus cycle can be lengthened by up to 1 μs. When the external memory is not being used, READY has no effect. Internal control of the number of wait states inserted into a bus cycle held not ready is available through configuration of CCR. READY has a weak internal pullup, so it goes to 1 unless externally pulled low. (Internal control of the number of wait states is not available on 8X9X parts). See Section 2.7.
HSI	Inputs to High Speed Input Unit. Four HSI pins are available: HSI.0, HSI.1, HSI.2, and HSI.3. Two of them (HSI.2 and HSI.3) are shared with the HSO Unit. The HSI pins are also used as inputs by EPROM parts in Programming mode. See Section 6.
HSO	Outputs from High Speed Output Unit. Six HSO pins are available: HSO.0, HSO.1, HSO.2, HSO.3, HSO.4, and HSO.5. Two of them (HSO.4 and HSO.5) are shared with the HSI Unit. See Section 7.
Port 0	8-bit high impedance input-only port. These pins can be used as digital inputs and/or as analog inputs to the on-chip A/D converter. These pins are also a mode input to EPROM parts in the Programming mode. See Section 10.
Port 1	8-bit quasi-bidirectional I/O port. See Section 10.
Port 2	8-bit multi-functional port. Six of its pins are shared with other functions in the 8096BH, the remaining 2 are quasi-bidirectional. These pins are also used to input and output control signals on EPROM parts in Programming Mode. See Section 10.
Ports 3 and 4	8-bit bi-directional I/O ports with open drain outputs. These pins are shared with the multiplexed address/data bus which has strong internal pullups. Ports 3 and 4 are also used as a command, address and data path by EPROM parts operating in the programming mode. See Sections 2.7 and 10.

14.2 Pin List

The following is a list of pins in alphabetical order. Where a pin has two names it has been listed under both names, except for the system bus pins, AD0–AD15, which are listed under Port 3 and Port 4.

Name	68-Pin PLCC	68-Pin PGA	48-Pin DIP
ACH0/P0.0	6	4	—
ACH1/P0.1	5	5	—
ACH2/P0.2	7	3	—
ACH3/P0.3	4	6	—
ACH4/P0.4/MOD.0	11	67	43
ACH5/P0.5/MOD.1	10	68	42
ACH6/P0.6/MOD.2	8	2	40
ACH7/P0.7/MOD.3	9	1	41
ALE/$\overline{\text{ADV}}$	62	16	34
ANGND	12	66	44
$\overline{\text{BHE}}$/$\overline{\text{WRH}}$	41	37	15
BUSWIDTH ($\overline{\text{TEST}}$)	64	14	—
CLKOUT	65	13	—
$\overline{\text{EA}}$	2	8	39
EXTINT/P2.2/$\overline{\text{PROG}}$	15	63	47
HSI.0	24	54	3
HSI.1	25	53	4
HSI.2/HSO.4	26	52	5
HSI.3/HSO.5	27	51	6
HSO.0	28	50	7
HSO.1	29	49	8
HSO.2	34	44	9
HSO.3	35	43	10
HSO.4/HSI.2	26	52	5
HSO.5/HSI.3	27	51	6
INST	63	15	—
NMI	3	7	—
PWM/P2.5/$\overline{\text{PDO}}$	39	39	13
PALE/P2.1/RXD	17	61	1
$\overline{\text{PROG}}$/P2.2/EXTNT	15	63	47
PVER/P2.0/TXD	18	60	2
P0.0/ACH0	6	4	—
P0.1/ACH1	5	5	—
P0.2/ACH2	7	3	—
P0.3/ACH3	4	6	—
P0.4/ACH4/MOD.0	11	67	43
PO.5/ACH5/MOD.1	10	68	42
PO.6/ACH6/MOD.2	8	2	40
PO.7/ACH7/MOD.3	9	1	41
P1.0	19	59	—
P1.1	20	58	—
P1.2	21	57	—
P1.3	22	56	—
P1.4	23	55	—
P1.5	30	48	—
P1.6	31	47	—
P1.7	32	46	—
P2.0/TXD/NER	18	60	2
P2.1/RXD/PALE	17	61	1
P2.2/EXTINT	15	63	47
P2.3/T2CLK	44	34	—
P2.4/T2RST	42	36	—
P2.5/PWM	39	39	13
P2.6	33	45	—
P2.7	38	40	—
P3.0/AD0	60	18	32
P3.1/AD1	59	19	31
P3.2/AD2	58	20	30
P3.3/AD3	57	21	29
P3.4/AD4	56	22	28
P3.5/AD5	55	23	27
P3.6/AD6	54	24	26
P3.7/AD7	53	25	25
P4.0/AD8	52	26	24
P4.1/AD9	51	27	23
P4.2/AD10	50	28	22
P4.3/AD11	49	29	21
P4.4/AD12	48	30	20
P4.5/AD13	47	31	19
P4.6/AD14	46	32	18
P4.7/AD15	45	33	17
$\overline{\text{RD}}$	61	17	33
READY	43	35	16
$\overline{\text{RESET}}$	16	62	48
RXD/P2.1	17	61	1
SALE/PVER/P2.0	18	60	2
SPROG/PD0/P2.5	39	39	13
TXD/P2.0	18	60	2
T2CLK/P2.3	44	34	—
T2RST/P2.4	42	36	—
VBB	37	41	12
VCC	1	9	38
VPD	14	64	46
VREF	13	65	45
VSS	68	10	11
VSS	36	42	37
$\overline{\text{WR}}$/$\overline{\text{WRL}}$	40	38	14
$\overline{\text{WRH}}$/$\overline{\text{BHE}}$	41	37	15
XTAL1	67	11	36
XTAL2	66	12	35

The Following pins are not bonded out in the 48-pin package:

P1.0 through P1.7, P0.0 through P0.3, P2.3, P2.4, P2.6, P2.7 CLKOUT, INST, NMI, $\overline{\text{TEST}}$, T2CLK (P2.3), T2RST (P2.4).

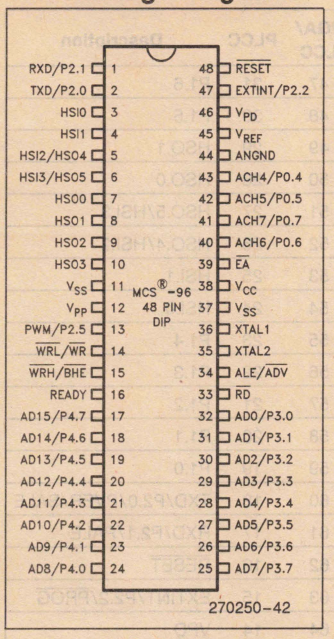
14.3 Packaging

The MCS-96 products are available in 48-pin and 68-pin packages, with and without A/D, and with and without on-chip ROM or EPROM. The MCS-96 numbering system is shown below. Section 14.4 shows the pinouts for the 48- and 68-pin packages. The 48-pin version is offered in a Dual-In-Line package while the 68-pin versions come in a Plastic Leaded Chip Carrier (PLCC), a Pin Grid Array (PGA) or a Type "B" Leadless Chip Carrier.

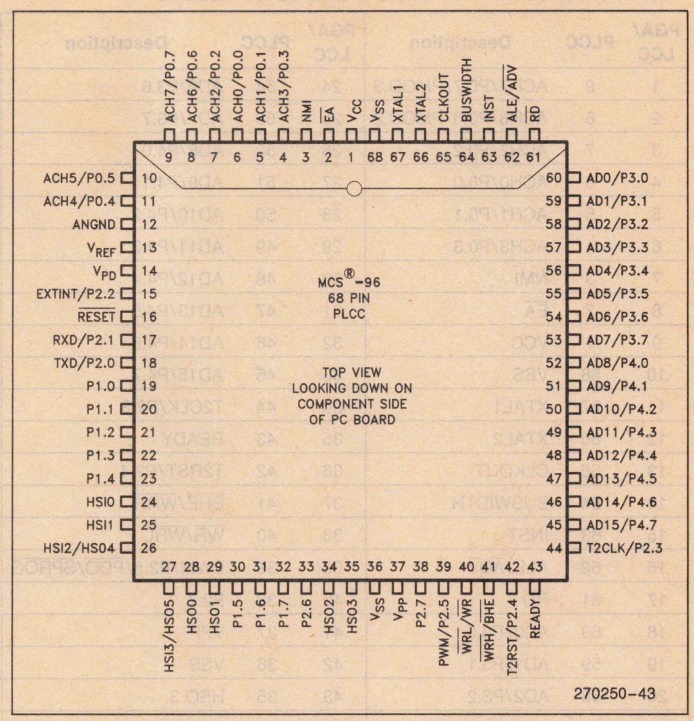
The MCS®-96 Family Nomenclature

		Without A/D	With A/D
ROMless 809XBH	48 Pin		C8095BH - Ceramic DIP P8095BH - Plastic DIP
	68 Pin	A8096BH - Ceramic PGA N8096BH - PLCC	A8097BH - Ceramic PGA N8097BH - PLCC
ROM 839XBH	48 Pin		C8395BH - Ceramic DIP P8395BH - Plastic DIP
	68 Pin	A8396BH - Ceramic PGA N8396BH - PLCC	A8397BH - Ceramic PGA N8397BH - PLCC
EPROM 879XBH	48 Pin		C8795BH - Ceramic DIP
	68 Pin	A8796BH - Ceramic PGA R8796BH - Ceramic LCC	A8797BH - Ceramic PGA R8797BH - Ceramic LCC
ROMless 8096	48 Pin		C8095-90 - Ceramic DIP P8095-90 - Plastic DIP
	68 Pin	A8096-90 - Ceramic PGA N8096-90 - PLCC	A8097-90 - Ceramic PGA N8097-90 - PLCC
ROM 8396	48 Pin		C8395-90 - Ceramic DIP P8395-90 - Plastic DIP
	68 Pin	A8396-90 - Ceramic PGA N8396-90 - PLCC	A8397-90 - Ceramic PGA N8397-90 - PLCC

14.4 Package Diagrams

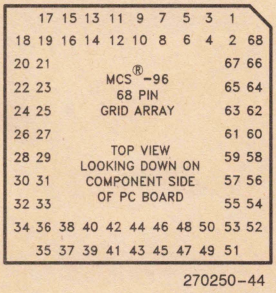


48-Pin Package

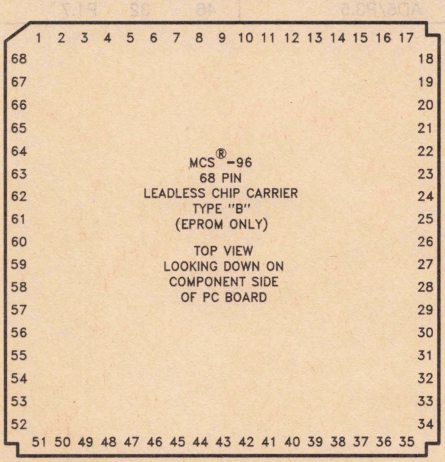


68-Pin Package (PLCC - Top View)

Pins Facing Down



68-Pin Package
(Pin Grid Array - Top View)

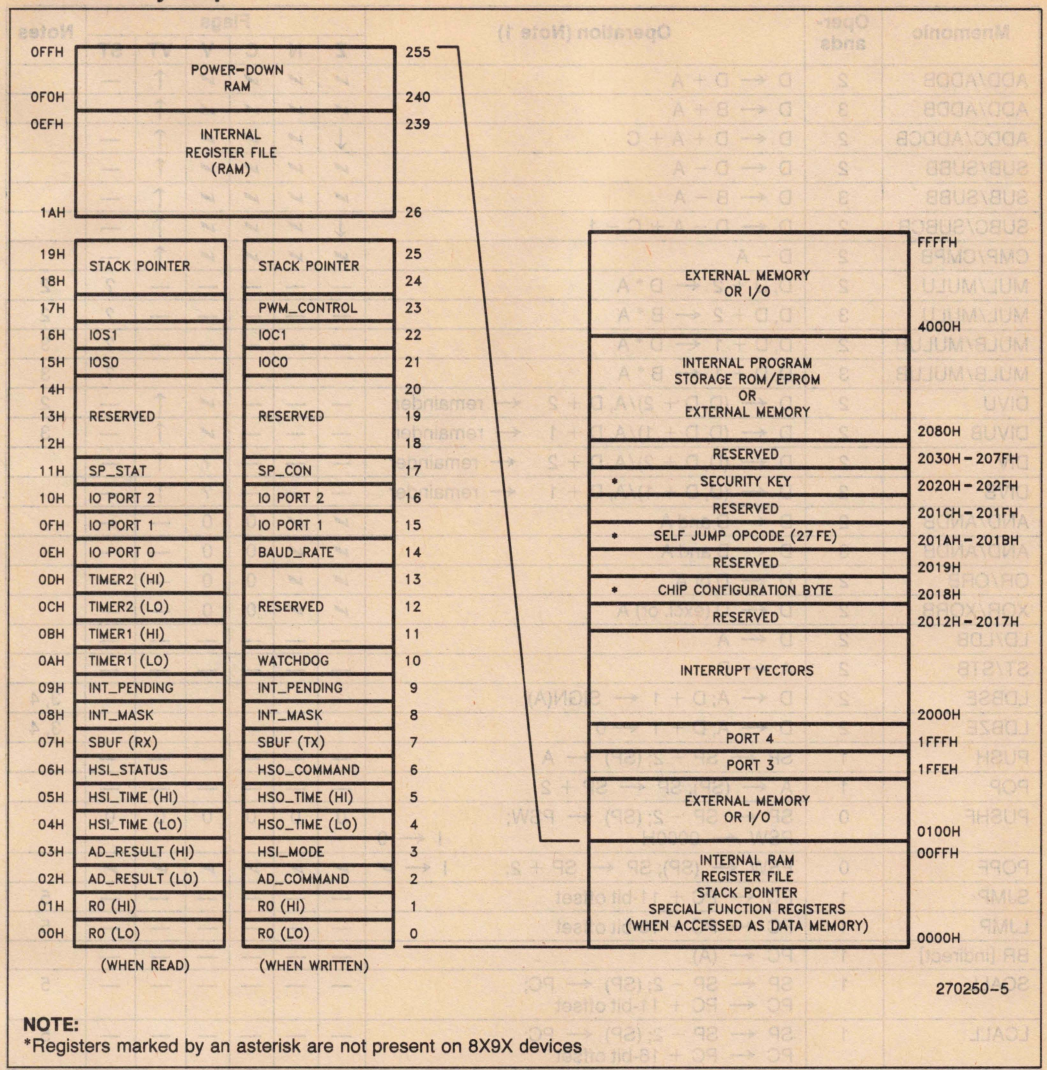


68-Pin Package (LCC - Top View)

14.5 PGA, PLCC and LCC Function Pinouts

PGA/ LCC	PLCC	Description	PGA/ LCC	PLCC	Description	PGA/ LCC	PLCC	Description
1	9	ACH7/P0.7/PMOD.3	24	54	AD6/P3.6	47	31	P1.6
2	8	ACH6/P0.6/PMOD.2	25	53	AD7/P3.7	48	30	P1.5
3	7	ACH2/P0.2	26	52	AD8/P4.0	49	29	HSO.1
4	6	ACH0/P0.0	27	51	AD9/P4.1	50	28	HSO.0
5	5	ACH1/P0.1	28	50	AD10/P4.2	51	27	HSO.5/HSI.3
6	4	ACH3/P0.3	29	49	AD11/P4.3	52	26	HSO.4/HSI.2
7	3	NMI	30	48	AD12/P4.4	53	25	HSI.1
8	2	$\overline{EA}$	31	47	AD13/P4.5	54	24	HSI.0
9	1	VCC	32	46	AD14/P4.6	55	23	P1.4
10	68	VSS	33	45	AD15/P4.7	56	22	P1.3
11	67	XTAL1	34	44	T2CLK/P2.3	57	21	P1.2
12	66	XTAL2	35	43	READY	58	20	P1.1
13	65	CLKOUT	36	42	T2RST/P2.4	59	19	P1.0
14	64	BUSWIDTH	37	41	$\overline{BHE}$/$\overline{WRH}$	60	18	TXD/P2.0/PVER/SALE
15	63	INST	38	40	$\overline{WR}$/$\overline{WRL}$	61	17	RXD/P2.1/PALE
16	62	ALE/$\overline{ADV}$	39	39	PWM/P2.5/$\overline{PDO}$/$\overline{SPROG}$	62	16	$\overline{RESET}$
17	61	$\overline{RD}$	40	38	P2.7	63	15	EXTINT/P2.2/$\overline{PROG}$
18	60	AD0/P3.0	41	37	VPP	64	14	VPD
19	59	AD1/P3.1	42	36	VSS	65	13	VREF
20	58	AD2/P3.2	43	35	HSO.3	66	12	ANGND
21	57	AD3/P3.3	44	34	HSO.2	67	11	ACH4/P0.4/PMOD.0
22	56	AD4/P3.4	45	33	P2.6	68	10	ACH5/P0.5/PMOD.1
23	55	AD5/P3.5	46	32	P1.7			

14.6 Memory Map



NOTE:
*Registers marked by an asterisk are not present on 8X9X devices

14.7 Instruction Summary

Mnemonic	Operands	Operation (Note 1)	Flags						Notes
			Z	N	C	V	VT	ST	
ADD/ADDB	2	D ← D + A	✓	✓	✓	✓	↑	—	
ADD/ADDB	3	D ← B + A	✓	✓	✓	✓	↑	—	
ADDC/ADDCB	2	D ← D + A + C	↓	✓	✓	✓	↑	—	
SUB/SUBB	2	D ← D − A	✓	✓	✓	✓	↑	—	
SUB/SUBB	3	D ← B − A	✓	✓	✓	✓	↑	—	
SUBC/SUBCB	2	D ← D − A + C − 1	↓	✓	✓	✓	↑	—	
CMP/CMPB	2	D − A	✓	✓	✓	✓	↑	—	
MUL/MULU	2	D, D + 2 ← D * A	—	—	—	—	—	?	2
MUL/MULU	3	D, D + 2 ← B * A	—	—	—	—	—	?	2
MULB/MULUB	2	D, D + 1 ← D * A	—	—	—	—	—	?	3
MULB/MULUB	3	D, D + 1 ← B * A	—	—	—	—	—	?	3
DIVU	2	D ← (D, D + 2)/A, D + 2 ← remainder	—	—	—	✓	↑	—	2
DIVUB	2	D ← (D, D + 1)/A, D + 1 ← remainder	—	—	—	✓	↑	—	3
DIV	2	D ← (D, D + 2)/A, D + 2 ← remainder	—	—	—	?	↑	—	
DIVB	2	D ← (D, D + 1)/A, D + 1 ← remainder	—	—	—	?	↑	—	
AND/ANDB	2	D ← D and A	✓	✓	0	0	—	—	
AND/ANDB	3	D ← B and A	✓	✓	0	0	—	—	
OR/ORB	2	D ← D or A	✓	✓	0	0	—	—	
XOR/XORB	2	D ← D (excl. or) A	✓	✓	0	0	—	—	
LD/LDB	2	D ← A	—	—	—	—	—	—	
ST/STB	2	A ← D	—	—	—	—	—	—	
LDBSE	2	D ← A; D + 1 ← SIGN(A)	—	—	—	—	—	—	3, 4
LDBZE	2	D ← A; D + 1 ← 0	—	—	—	—	—	—	3, 4
PUSH	1	SP ← SP − 2; (SP) ← A	—	—	—	—	—	—	
POP	1	A ← (SP); SP ← SP + 2	—	—	—	—	—	—	
PUSHF	0	SP ← SP − 2; (SP) ← PSW; PSW ← 0000H I ← 0	0	0	0	0	0	0	
POPF	0	PSW ← (SP); SP ← SP + 2; I ← ✓	✓	✓	✓	✓	✓	✓	
SJMP	1	PC ← PC + 11-bit offset	—	—	—	—	—	—	5
LJMP	1	PC ← PC + 16-bit offset	—	—	—	—	—	—	5
BR [indirect]	1	PC ← (A)	—	—	—	—	—	—	
SCALL	1	SP ← SP − 2; (SP) ← PC; PC ← PC + 11-bit offset	—	—	—	—	—	—	5
LCALL	1	SP ← SP − 2; (SP) ← PC; PC ← PC + 16-bit offset	—	—	—	—	—	—	5
RET	0	PC ← (SP); SP ← SP + 2	—	—	—	—	—	—	
J (conditional)	1	PC ← PC + 8-bit offset (if taken)	—	—	—	—	—	—	5
JC	1	Jump if C = 1	—	—	—	—	—	—	5
JNC	1	Jump if C = 0	—	—	—	—	—	—	5
JE	1	Jump if Z = 1	—	—	—	—	—	—	5

NOTES:
1. If the mnemonic ends in "B", a byte operation is performed, otherwise a word operation is done. Operands D, B, and A must conform to the alignment rules for the required operand type. D and B are locations in the Register File; A can be located anywhere in memory.
2. D, D + 2 are consecutive WORDS in memory; D is DOUBLE-WORD aligned.
3. D, D + 1 are consecutive BYTES in memory; D is WORD aligned.
4. Changes a byte to a word.
5. Offset is a 2's complement number.

Mnemonic	Operands	Operation (Note 1)	Flags						Notes
			Z	N	C	V	VT	ST	
JNE	1	Jump if Z = 0	—	—	—	—	—	—	5
JGE	1	Jump if N = 0	—	—	—	—	—	—	5
JLT	1	Jump if N = 1	—	—	—	—	—	—	5
JGT	1	Jump if N = 0 and Z = 0	—	—	—	—	—	—	5
JLE	1	Jump if N = 1 or Z = 1	—	—	—	—	—	—	5
JH	1	Jump if C = 1 and Z = 0	—	—	—	—	—	—	5
JNH	1	Jump if C = 0 or Z = 1	—	—	—	—	—	—	5
JV	1	Jump if V = 1	—	—	—	—	—	—	5
JNV	1	Jump if V = 0	—	—	—	—	—	—	5
JVT	1	Jump if VT = 1; Clear VT	—	—	—	—	0	—	5
JNVT	1	Jump if VT = 0; Clear VT	—	—	—	—	0	—	5
JST	1	Jump if ST = 1	—	—	—	—	—	—	5
JNST	1	Jump if ST = 0	—	—	—	—	—	—	5
JBS	3	Jump if Specified Bit = 1	—	—	—	—	—	—	5, 6
JBC	3	Jump if Specified Bit = 0	—	—	—	—	—	—	5, 6
DJNZ	1	D ← D − 1; if D ≠ 0 then PC ← PC + 8-bit offset	—	—	—	—	—	—	5
DEC/DECB	1	D ← D − 1	↙	↙	↙	↙	↑	—	
NEG/NEGB	1	D ← 0 − D	↙	↙	↙	↙	↑	—	
INC/INCB	1	D ← D + 1	↙	↙	↙	↙	↑	—	
EXT	1	D ← D; D + 2 ← Sign (D)	↙	↙	0	0	—	—	2
EXTB	1	D ← D; D + 1 ← Sign(D)	↙	↙	0	0	—	—	3
NOT/NOTB	1	D ← Logical Not (D)	↙	↙	0	0	—	—	
CLR/CLRB	1	D ← 0	1	0	0	0	—	—	
SHL/SHLB/SHLL	2	C ← msb — — — — — lsb ← 0	↙	?	↙	↙	↑	—	7
SHR/SHRB/SHRL	2	0 → msb — — — — — lsb → C	↙	?	↙	0	—	↙	7
SHRA/SHRAB/SHRAL	2	msb → msb — — — — — lsb → C	↙	↙	↙	0	—	↙	7
SETC	0	C ← 1	—	—	1	—	—	—	
CLRC	0	C ← 0	—	—	0	—	—	—	
CLRVT	0	VT ← 0	—	—	—	—	0	—	
RST	0	PC ← 2080H	0	0	0	0	0	0	8
DI	0	Disable All Interrupts (I ← 0)	—	—	—	—	—	—	
EI	0	Enable All Interrupts (I ← 1)	—	—	—	—	—	—	
NOP	0	PC ← PC + 1	—	—	—	—	—	—	
SKIP	0	PC ← PC + 2	—	—	—	—	—	—	
NORML	2	Left shift till msb = 1; D ← shift count	↙	?	0	—	—	—	7
TRAP	0	SP ← SP − 2; (SP) ← PC PC ← (2010H)	—	—	—	—	—	—	9

NOTES:

1. If the mnemonic ends in "B", a byte operation is performed, otherwise a word operation is done. Operands D, B and A must conform to the alignment rules for the required operand type. D and B are locations in the Register File; A can be located anywhere in memory.
5. Offset is a 2's complement number.
6. Specified bit is one of the 2048 bits in the register file.
7. The "L" (Long) suffix indicates double-word operation.
8. Initiates a Reset by pulling $\overline{\text{RESET}}$ low. Software should re-initialize all the necessary registers with code starting at 2080H.
9. The assembler will not accept this mnemonic.

14.8 Opcode and State Time Listing

MNEMONIC	OPERANDS	DIRECT			IMMEDIATE			INDIRECT⑧ NORMAL			INDIRECT⑧ AUTO-INC.		INDEXED⑧ SHORT			INDEXED⑧ LONG	
		OPCODE	BYTES	STATE TIMES	OPCODE	BYTES	STATE TIMES	OPCODE	BYTES	STATE① TIMES	BYTES	STATE① TIMES	OPCODE	BYTES	STATE① TIMES⑧	BYTES	STATE① TIMES⑧
ARITHMETIC INSTRUCTIONS																	
ADD	2	64	3	4	65	4	5	66	3	6/11	3	7/12	67	4	6/11	5	7/12
ADD	3	44	4	5	45	5	6	46	4	7/12	4	8/13	47	5	7/12	6	8/13
ADDB	2	74	3	4	75	3	4	76	3	6/11	3	7/12	77	4	6/11	5	7/12
ADDB	3	54	4	5	55	4	5	56	4	7/12	4	8/13	57	5	7/12	6	8/13
ADDC	2	A4	3	4	A5	4	5	A6	3	6/11	3	7/12	A7	4	6/11	5	7/12
ADDCB	2	B4	3	4	B5	3	4	B6	3	6/11	3	7/12	B7	4	6/11	5	7/12
SUB	2	68	3	4	69	4	5	6A	3	6/11	3	7/12	6B	4	6/11	5	7/12
SUB	3	48	4	5	49	5	6	4A	4	7/12	4	8/13	4B	5	7/12	6	8/13
SUBB	2	78	3	4	79	3	4	7A	3	6/11	3	7/12	7B	4	6/11	5	7/12
SUBB	3	58	4	5	59	4	5	5A	4	7/12	4	8/13	5B	5	7/12	6	8/13
SUBC	2	A8	3	4	A9	4	5	AA	3	6/11	3	7/12	AB	4	6/11	5	7/12
SUBCB	2	B8	3	4	B9	3	4	BA	3	6/11	3	7/12	BB	4	6/11	5	7/12
CMP	2	88	3	4	89	4	5	8A	3	6/11	3	7/12	8B	4	6/11	5	7/12
CMPB	2	98	3	4	99	3	4	9A	3	6/11	3	7/12	9B	4	6/11	5	7/12
MULU	2	6C	3	25	6D	4	26	6E	3	27/32	3	28/33	6F	4	27/32	5	28/33
MULU	3	4C	4	26	4D	5	27	4E	4	28/33	4	29/34	4F	5	28/33	6	29/34
MULUB	2	7C	3	17	7D	3	17	7E	3	19/24	3	20/25	7F	4	19/24	5	20/25
MULUB	3	5C	4	18	5D	4	18	5E	4	20/25	4	21/26	5F	5	20/25	6	21/26
MUL	2	②	4	29	②	5	30	②	4	31/36	4	32/37	②	5	31/36	6	32/37
MUL	3	②	5	30	②	6	31	②	5	32/37	5	33/38	②	6	32/37	7	33/38
MULB	2	②	4	21	②	4	21	②	4	23/28	4	24/29	②	5	23/28	6	24/29
MULB	3	②	5	22	②	5	22	②	5	24/29	5	25/30	②	6	24/29	7	25/30
DIVU	2	8C	3	25	8D	4	26	8E	3	28/32	3	29/33	8F	4	28/32	5	29/33
DIVUB	2	9C	3	17	9D	3	17	9E	3	20/24	3	21/25	9F	4	20/24	5	21/25
DIV	2	②	4	29	②	5	30	②	4	32/36	4	33/37	②	5	32/36	6	33/37
DIVB	2	②	4	21	②	4	21	②	4	24/28	4	25/29	②	5	24/28	6	25/29

270250-46

NOTES:

*Long indexed and Indirect + instructions have identical opcodes with Short indexed and Indirect modes, respectively. The second byte of instructions using any Indirect or indexed addressing mode specifies the exact mode used. If the second byte is even, use Indirect or Short indexed. If it is odd, use Indirect + or Long indexed. In all cases the second byte of the instruction always specifies an even (word) location for the address referenced.

① Number of state times shown for internal/external operands.

② The opcodes for signed multiply and divide are the opcodes for the unsigned functions with an "FE" appended as a prefix.

⑧ State times shown for 16-bit bus.

MNEMONIC	OPERANDS	DIRECT			IMMEDIATE			INDIRECT⑧ NORMAL			INDIRECT⑧ AUTO-INC.		INDEXED⑧ SHORT			INDEXED⑧ LONG	
		OPCODE	BYTES	STATE TIMES	OPCODE	BYTES	STATE TIMES	OPCODE	BYTES	STATE① TIMES	BYTES	STATE① TIMES	OPCODE	BYTES	STATE① TIMES⑧	BYTES	STATE① TIMES⑧
LOGICAL INSTRUCTIONS																	
AND	2	60	3	4	61	4	5	62	3	6/11	3	7/12	63	4	6/11	5	7/12
AND	3	40	4	5	41	5	6	42	4	7/12	4	8/13	43	5	7/12	6	8/13
ANDB	2	70	3	4	71	3	4	72	3	6/11	3	7/12	73	4	6/11	5	7/12
ANDB	3	50	4	5	51	4	5	52	4	7/12	4	8/13	53	5	7/12	6	8/13
OR	2	80	3	4	81	4	5	82	3	6/11	3	7/12	83	4	6/11	5	7/12
ORB	2	90	3	4	91	3	4	92	3	6/11	3	7/12	93	4	6/11	5	7/12
XOR	2	84	3	4	85	4	5	86	3	6/11	3	7/12	87	4	6/11	5	7/12
XORB	2	94	3	4	95	3	4	96	3	6/11	3	7/12	97	4	6/11	5	7/12
DATA TRANSFER INSTRUCTIONS																	
LD	2	A0	3	4	A1	4	5	A2	3	6/11	3	7/12	A3	4	6/11	5	7/12
LDB	2	B0	3	4	B1	3	4	B2	3	6/11	3	7/12	B3	4	6/11	5	7/12
ST	2	C0	3	4	—	—	—	C2	3	7/11	3	8/12	C3	4	7/11	5	8/12
STB	2	C4	3	4	—	—	—	C6	3	7/11	3	8/12	C7	4	7/11	5	8/12
LDBSE	2	BC	3	4	BD	3	4	BE	3	6/11	3	7/12	BF	4	6/11	5	7/12
LDBZE	2	AC	3	4	AD	3	4	AE	3	6/11	3	7/12	AF	4	6/11	5	7/12
STACK OPERATIONS (internal stack)																	
PUSH	1	C8	2	8	C9	3	8	CA	2	11/15	2	12/16	CB	3	11/15	4	12/16
POP	1	CC	2	12	—	—	—	CE	2	14/18	2	14/18	CF	3	14/18	4	14/18
PUSHF	0	F2	1	8													
POPF	0	F3	1	9													
STACK OPERATIONS (external stack)																	
PUSH	1	C8	2	12	C9	3	12	CA	2	15/19	2	16/20	CB	3	15/19	4	16/20
POP	1	CC	2	14	—	—	—	CE	2	16/20	2	16/20	CF	3	16/20	4	16/20
PUSHF	0	F2	1	12													
POPF	0	F3	1	13													

JUMPS AND CALLS							
MNEMONIC	OPCODE	BYTES	STATES	MNEMONIC	OPCODE	BYTES	STATES
LJMP	E7	3	8	LCALL	EF	3	13/16⑤
SJMP	20-27④	2	8	SCALL	28-2F④	2	13/16⑤
BR[ ]	E3	2	8	RET	F0	1	12/16⑤
				TRAP③	F7	1	21/24

270250-47

NOTES:
① Number of state times shown for internal/external operands.
③ The assembler does not accept this mnemonic.
④ The least significant 3 bits of the opcode are concatenated with the following 8 bits to form an 11-bit, 2's complement, offset for the relative call or jump.
⑤ State times for stack located internal/external.
⑧ State times shown for 16-bit bus.

CONDITIONAL JUMPS

All conditional jumps are 2 byte instructions. They require 8 state times if the jump is taken, 4 if it is not.(8)

MNEMONIC	OPCODE	MNEMONIC	OPCODE	MNEMONIC	OPCODE	MNEMONIC	OPCODE
JC	DB	JE	DF	JGE	D6	JGT	D2
JNC	D3	JNE	D7	JLT	DE	JLE	DA
JH	D9	JV	DD	JVT	DC	JST	D8
JNH	D1	JNV	D5	JNVT	D4	JNST	D0

JUMP ON BIT CLEAR OR BIT SET

These instructions are 3-byte instructions. They require 9 state times if the jump is taken, 5 if it is not.(8)

	BIT NUMBER							
MNEMONIC	0	1	2	3	4	5	6	7
JBC	30	31	32	33	34	35	36	37
JBS	38	39	3A	3B	3C	3D	3E	3F

LOOP CONTROL

MNEMONIC	OPCODE	BYTES	STATE TIMES
DJNZ	E0	3	5/9 STATE TIME (NOT TAKEN/TAKEN)(8)

SINGLE REGISTER INSTRUCTIONS

MNEMONIC	OPCODE	BYTES	STATES(8)	MNEMONIC	OPCODE	BYTES	STATES(8)
DEC	05	2	4	EXT	06	2	4
DECB	15	2	4	EXTB	16	2	4
NEG	03	2	4	NOT	02	2	4
NEGB	13	2	4	NOTB	12	2	4
INC	07	2	4	CLR	01	2	4
INCB	17	2	4	CLRB	11	2	4

SHIFT INSTRUCTIONS

INSTR MNEMONIC	WORD		INSTR MNEMONIC	BYTE		INSTR MNEMONIC	DBL WD		STATE TIMES(8)
	OP	B		OP	B		OP	B	
SHL	09	3	SHLB	19	3	SHLL	0D	3	7 + 1 PER SHIFT(7)
SHR	08	3	SHRB	18	3	SHRL	0C	3	7 + 1 PER SHIFT(7)
SHRA	0A	3	SHRAB	1A	3	SHRAL	0E	3	7 + 1 PER SHIFT(7)

SPECIAL CONTROL INSTRUCTIONS

MNEMONIC	OPCODE	BYTES	STATES(8)	MNEMONIC	OPCODE	BYTES	STATES(8)
SETC	F9	1	4	DI	FA	1	4
CLRC	F8	1	4	EI	FB	1	4
CLRVT	FC	1	4	NOP	FD	1	4
RST(6)	FF	1	166	SKIP	00	2	4

NORMALIZE

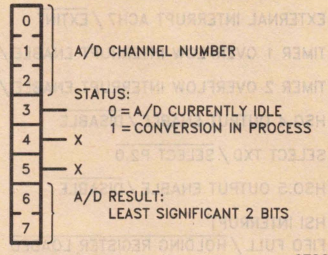
MNEMONIC	OPCODE	BYTES	STATE TIMES
NORML	0F	3	11 + 1 PER SHIFT

NOTES:

6. This instruction takes 2 states to pull $\overline{\text{RESET}}$ low, then holds it low for 2 states to initiate a reset. The reset takes 12 states, at which time the program restarts at location 2080H. If a capacitor is tied to $\overline{\text{RESET}}$, the pin may take longer to go low and may never reach the V_{OL} specification.
7. Execution will take at least 8 states, even for 0 shift.
8. State times shown for 16-bit bus.

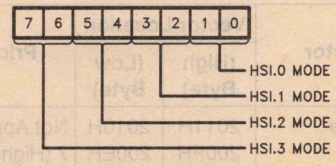
14.9 SFR Summary

A/D Result LO (02H)



270250-48

HSI_Mode (03H)

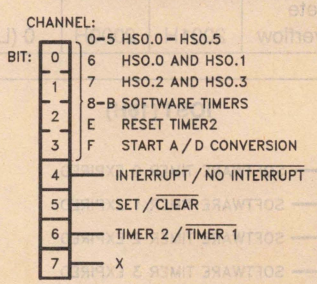


WHERE EACH 2-BIT MODE CONTROL FIELD DEFINES ONE OF 4 POSSIBLE MODES:

00 8 POSITIVE TRANSITIONS
01 EACH POSITIVE TRANSITION
10 EACH NEGATIVE TRANSITION
11 EVERY TRANSITION (POSITIVE AND NEGATIVE)

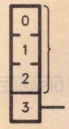
270250-49

HSO Command (06H)



270250-50

A/D Command (02H)

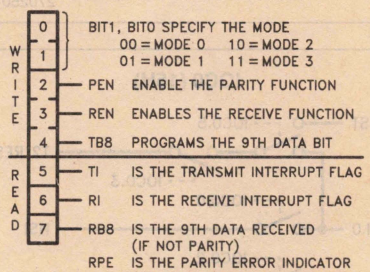


CHANNEL # SELECTS WHICH OF THE 8 ANALOG INPUT CHANNELS IS TO BE CONVERTED TO DIGITAL FORM.

GO INDICATES WHEN THE CONVERSION IS TO BE INITIATED (GO = 1 MEANS START NOW, GO = 0 MEANS THE CONVERSION IS TO BE INITIATED BY THE HSO UNIT AT A SPECIFIED TIME).

270250-51

SPCON/SPSTAT (11H)



270250-52

Baud Rate Calculations

Using XTAL1:

$$\text{Mode 0: } \frac{\text{Baud}}{\text{Rate}} = \frac{\text{XTAL1 frequency}}{4*(B+1)};\ B \neq 0$$

$$\text{Others: } \frac{\text{Baud}}{\text{Rate}} = \frac{\text{XTAL1 frequency}}{64*(B+1)}$$

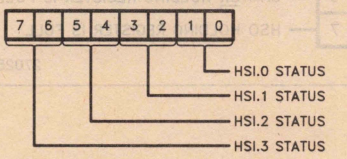
Using T2CLK:

$$\text{Mode 0: } \frac{\text{Baud}}{\text{Rate}} = \frac{\text{T2CLK frequency}}{B};\ B \neq 0$$

$$\text{Others: } \frac{\text{Baud}}{\text{Rate}} = \frac{\text{T2CLK frequency}}{16*B};\ B \neq 0$$

Note that B cannot equal 0, except when using XTAL1 in other than Mode 0.

HSI_Status (06H)



WHERE FOR EACH 2-BIT STATUS FIELD THE LOWER BIT INDICATES WHETHER OR NOT AN EVENT HAS OCCURED ON THIS PIN AND THE UPPER BIT INDICATES THE CURRENT STATUS OF THE PIN.

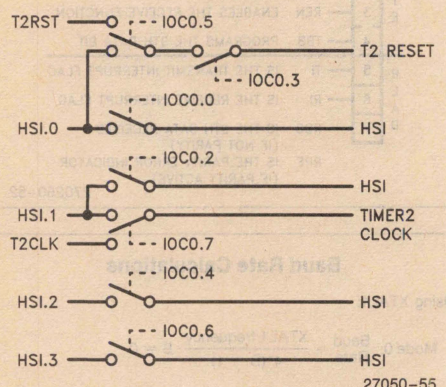
270250-53

IOC0 (15H)

0	HSI.0 INPUT ENABLE / $\overline{\text{DISABLE}}$
1	TIMER 2 RESET EACH WRITE
2	HSI.1 INPUT ENABLE / $\overline{\text{DISABLE}}$
3	TIMER 2 EXTERNAL RESET ENABLE / $\overline{\text{DISABLE}}$
4	HSI.2 INPUT ENABLE / $\overline{\text{DISABLE}}$
5	TIMER 2 RESET SOURCE HSI.0 / $\overline{\text{T2RST}}$
6	HSI.3 INPUT ENABLE / $\overline{\text{DISABLE}}$
7	TIMER 2 CLOCK SOURCE HSI.1 / $\overline{\text{T2CLK}}$

270250-54

IOC0 (15H)



27050-55

IOS0 (15H)

0	HSO.0 CURRENT STATE
1	HSO.1 CURRENT STATE
2	HSO.2 CURRENT STATE
3	HSO.3 CURRENT STATE
4	HSO.4 CURRENT STATE
5	HSO.5 CURRENT STATE
6	CAM OR HOLDING REGISTER IS FULL
7	HSO HOLDING REGISTER IS FULL

270250-56

IOC1 (16H)

0	SELECT PWM / $\overline{\text{SELECT P2.5}}$
1	EXTERNAL INTERRUPT ACH7 / $\overline{\text{EXTINT}}$
2	TIMER 1 OVERFLOW INTERRUPT ENABLE / $\overline{\text{DISABLE}}$
3	TIMER 2 OVERFLOW INTERRUPT ENABLE / $\overline{\text{DISABLE}}$
4	HSO.4 OUTPUT ENABLE / $\overline{\text{DISABLE}}$
5	SELECT TXD / $\overline{\text{SELECT P2.0}}$
6	HSO.5 OUTPUT ENABLE / $\overline{\text{DISABLE}}$
7	HSI INTERRUPT FIFO FULL / $\overline{\text{HOLDING REGISTER LOADED}}$

270250-57

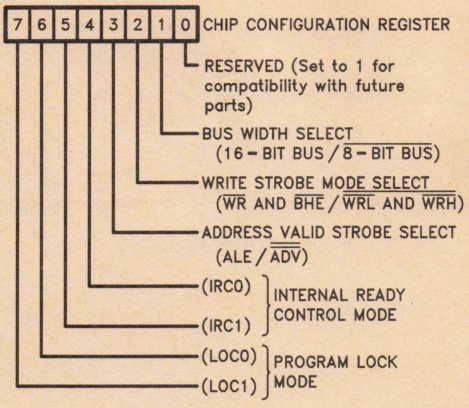
Vector	Vector Location (High Byte)	Vector Location (Low Byte)	Priority
Software	2011H	2010H	Not Applicable
Extint	200FH	200EH	7 (Highest)
Serial Port	200DH	200CH	6
Software Timers	200BH	200AH	5
HSI.0	2009H	2008H	4
High Speed Outputs	2007H	2006H	3
HSI Data Available	2005H	2004H	2
A/D Conversion Complete	2003H	2002H	1
Timer Overflow	2001H	2000H	0 (Lowest)

IOS1 (16H)

0	SOFTWARE TIMER 0 EXPIRED
1	SOFTWARE TIMER 1 EXPIRED
2	SOFTWARE TIMER 2 EXPIRED
3	SOFTWARE TIMER 3 EXPIRED
4	TIMER 2 HAS OVERFLOW
5	TIMER 1 HAS OVERFLOW
6	HSI FIFO IS FULL
7	HSI HOLDING REGISTER DATA AVAILABLE

270250-58

Chip Configuration



Internal Ready Control

IRC1	IRC0	Description
0	0	Limit to 1 Wait State
0	1	Limit to 2 Wait States
1	0	Limit to 3 Wait States
1	1	Disable Internal Ready Control

Program Lock Modes

LOC1	LOC0	Protection
0	0	Read and Write Protected
0	1	Read Protected
1	0	Write Protected
1	1	No Protection

Programming Function PMODE Values

PMODE	Programming Mode
0–4	Reserved
5	Slave Programming
6–0BH	Reserved
0CH	Auto Programming Mode
0DH	Program Configuration Byte
0EH–0FH	Reserved

Slave Programming Mode Commands

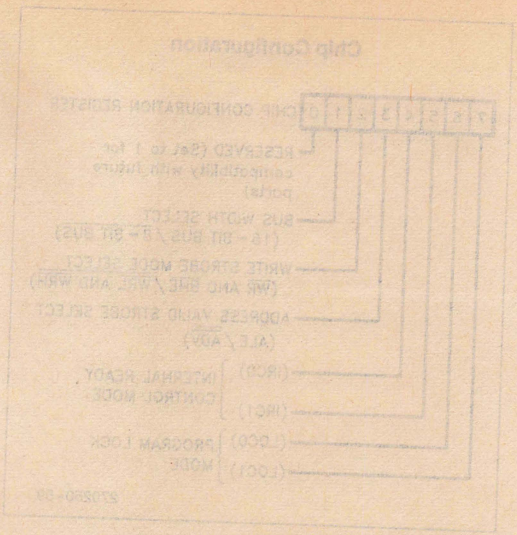
P4.7	P4.6	Action
0	0	Word Dump
0	1	Data Verify
1	0	Data Program
1	1	Reserved

8X9XBH Signature Word

Device	Signature Word
879XBH	896FH
839XBH	896EH
809XBH	Undefined

Port 2 Pin Functions

Port	Function	Alternate Function
P2.0	Output	TXD (Serial Port Transmit)
P2.1	Input	RXD (Serial Port Receive)
P2.2	Input	EXTINT (External Interrupt)
P2.3	Input	T2CLK (Timer 2 Clock)
P2.4	Input	T2RST (Timer 2 Reset)
P2.5	Output	PWM (Pulse Width Modulation)



Internal Ready Control

IRC1	IRC0	Description
0	0	Limit to 1 Wait State
0	1	Limit to 2 Wait States
1	0	Limit to 3 Wait States
1	1	Disable Internal Ready Control

Program Lock Modes

LOCK1	LOCK0	Protection
0	0	Read and Write Protected
0	1	Read Protected
1	0	Write Protected
1	1	No Protection

MCS®-96 Instruction Set

13

MCS®-96 INSTRUCTION SET

OVERVIEW

This chapter of the manual gives a description of each instruction recognized by the 8096. The instructions are sorted alphabetically by the mnemonic used in the assembly language for the 8096. A summary of the instruction set is included in Section 14 of the MCS®-96 Architecture chapter.

The instruction set descriptions in the following sections do not always show the effect on the program counter (PC). Unless otherwise specified, all instructions increment the PC by the number of bytes in the instruction.

A set of acronyms are used to make the instruction set descriptions easier to read, their definitions are listed below:

aa. A two bit field within an opcode which selects the basic addressing mode user. This field is only present in those opcodes which allow address mode options. The encoding of the field is as follows:

aa	Addressing mode
00	Register direct
01	Immediate
10	Indirect
11	Indexed

The selection between indirect and indirect with autoincrement or between short and long indexing is done based on the least significant bit of the instruction byte which follows the opcode. This type selects the 16-bit register which is to take part in the address calculation. Since the 8096 requires that words be aligned on even byte boundaries this bit would be otherwise unused.

breg. A byte register in the internal register file. When confusion could exist as to whether this field refers to a source or a destination register it will be prefixed with an "S" or a "D".

baop. A byte operand which is addressed by any of the address modes discussed in Section 3.2 of the MCS-96 Architecture chapter.

bitno. A three bit field within an instruction op-code which selects one of the eight bits in a byte.

wreg. A word register in the internal register file. When confusion could exist as to whether this field refers to a source register or a destination register it will be prefixed with an "S" or a "D".

waop. A word operand which is addressed by any of the address modes discussed in Section 3.2 of the MCS-96 Architecture chapter.

Lreg. A 32-bit register in the internal register file.

BEA. Extra bytes of code required for the address mode selected.

CEA. Extra state times (cycles) required for the address mode selected.

cadd. An address in the program code.

Flag Settings. The modification to the flag setting is shown for each instruction. A checkmark (✓) means that the flag is set or cleared as appropriate. A hyphen means that the flag is not modified. A one or zero (1) or (0) indicates that the flag will be in that state after the instruction. An up arrow (↑) indicates that the instruction may set the flag if it is appropriate but will not clear the flag. A down arrow (↓) indicates that the flag can be cleared but not set by the instruction. A question mark (?) indicates that the flag will be left in an indeterminant state after the operation.

Generic Jumps and Calls. The assembler for the MCS-96 family provides for generic jumps and calls. For all of the conditional jump instructions a "B" can be substituted for the "J" and the assembler will generate a code sequence which is logically equivalent but can reach anywhere in the memory. A JH can only jump about 128 locations from the current program counter; a BH can jump anywhere in memory. In a like manner a BR will cause a SJMP or LJMP to be generated as appropriate and a CALL will cause a SCALL or LCALL to be generated. The assembler user's guide should be consulted for the algorithms used by the assembler to convert these generic instructions into actual machine instructions.

Indirect Shifts. The indirect shift operations use registers 24 through 255 (18H–0FFH), since 0–15 are direct operators and registers 16 through 23 are Special Function Registers. Note that indirect shifts through SFRs are illegal operations.

The maximum shift count is 31 (1FH). Count values above this will be truncated to the 5 least significant bits.

1. ADD (Two Operands) — ADD WORDS

Operation: The sum of the two word operands is stored into the destination (leftmost) operand.

(DEST) ← (DEST) + (SRC)

Assembly Language Format:

	DST	SRC
ADD	wreg,	waop

Object Code Format: [011001aa] [waop] [wreg]

Bytes: 2 + BEA
States: 4 + CEA

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	—

2. ADD (Three Operands) — ADD WORDS

Operation: The Sum of the second and third word operands is stored into the destination (leftmost) operand.

(DEST) ← (SRC1) + (SRC2)

Assembly Language Format:

	DST	SRC1	SRC2
ADD	Dwreg,	Swreg,	waop

Object Code Format: [010001aa] [waop] [Swreg] [Dwreg]

Bytes: 3 + BEA
States: 5 + CEA

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	—

3. ADDB (Two Operands) — ADD BYTES

Operation: The sum of the two byte operands is stored into the destination (leftmost) operand.

(DEST) ← (DEST) + (SRC)

Assembly Language Format:

	DST	SRC
ADDB	breg,	baop

Object Code Format: [011101aa] [baop] [breg]

Bytes 2 + BEA
States: 4 + CEA

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	—

4. ADDB (Three Operands) — ADD BYTES

Operation: The sum of the second and third byte operands is stored into the destination (leftmost) operand.

(DEST) ← (SRC1) + (SRC2)

Assembly Language Format:

	DST	SRC1	SRC2
ADDB	Dbreg,	Sbreg,	baop

Object Code Format: [010101aa] [baop] [Sbreg] [Dbreg]

Bytes: 3 + BEA
States 5 + CEA

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	—

5. ADDC — ADD WORDS WITH CARRY

Operation: The sum of the two word operands and the carry flag (0 or 1) is stored into the destination (leftmost) operand.

(DEST) ← (DEST) + (SRC) + C

Assembly Language Format:

	DST	SRC
ADDC	wreg,	waop

Object Code Format: [101001aa] [waop] [wreg]

Bytes: 2 + BEA
States: 4 + BEA

Flags Affected					
Z	N	C	V	VT	ST
↓	✓	✓	✓	↑	—

6. ADDCB — ADD BYTES WITH CARRY

Operation: The sum of the two byte operands and the carry flag (0 or 1) is stored into the destination (leftmost) operand.

(DEST) ← (DEST) + (SRC) + C

Assembly Language Format:

	DST	SRC
ADDCB	breg,	baop

Object Code Format: [101101aa] [baop] [breg]

Bytes: 2 + BEA
States: 4 + CEA

Flags Affected					
Z	N	C	V	VT	ST
↓	✓	✓	✓	↑	—

7. AND (Two Operands) — LOGICAL AND WORDS

Operation: The two word operands are ANDed, the result having a 1 only in those bit positions where both operands had a 1, with zeroes in all other bit positions. The result is stored into the destination (leftmost) operand.

(DEST) ← (DEST) AND (SRC)

Assembly Language Format:

```
        DST     SRC
AND     wreg,   waop
```

Object Code Format: [011000aa] [waop] [wreg]

Bytes: 2 + BEA
States 4 + CEA

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	0	0	—	—

8. AND (Three Operands) — LOGICAL AND WORDS

Operation: The second and third word operands are ANDed, the result having a 1 only in those bit positions where both operands had a 1, with zeroes in all other bit positions. The result is stored into the destination (leftmost) operand.

(DEST) ← (SRC1) AND (SRC2)

Assembly Language Format:

```
        DST     SRC1    SRC2
AND     Dwreg,  Swreg,  waop
```

Object Code Format: [010000aa] [waop] [Swreg] [Dwreg]

Bytes: 3 + BEA
States: 5 + CEA

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	0	0	—	—

9. ANDB (Two Operands) — LOGICAL AND BYTES

Operation: The two byte operands are ANDed, the result having a 1 only in those bit positions where both operands had a 1, with zeroes in all other bit positions. The result is stored into the destination (leftmost) operand.

(DEST) ← (DEST) AND (SRC)

Assembly Language Format:

```
        DST     SRC
ANDB    breg,   baop
```

Object Code Format: [011100aa] [baop] [breg]

Bytes: 2 + BEA
States: 4 + CEA

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	0	0	—	—

10. ANDB (Three Operands) — LOGICAL AND BYTES

Operation: The second and third byte operands are ANDed, the result having a 1 only in those bit positions where both operands had a 1, with zeroes in all other bit positions. The result is stored into the destination (leftmost) operand.

(DEST) ← (SRC1) AND (SRC2)

Assembly Language Format:

```
        DST     SRC1    SRC2
ANDB    Dbreg,  Sbreg,  baop
```

Object Code Format: [010100aa] [baop] [Sbreg] [Dbreg]

Bytes: 3 + BEA
States: 5 + CEA

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	0	0	—	—

11. BR (Indirect) — BRANCH INDIRECT

Operation: The execution continues at the address specified in the operand word register.

PC ← (DEST)

Assembly Language Format: BR [wreg]

Object Code Format: [11100011] [wreg]

Bytes: 2
States: 8

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

12. CLR — CLEAR WORD

Operation: The value of the word operand is set to zero.

(DEST) ← 0

Assembly Language Format: CLR wreg

Object Code Format: [00000001] [wreg]

Bytes: 2
States: 4

Flags Affected					
Z	N	C	V	VT	ST
1	0	0	0	—	—

13. CLRB — CLEAR BYTE

Operation: The value of the byte operand is set to zero.
(DEST) ← 0

Assembly Language Format: CLRB breg

Object Code Format: [00010001] [breg]

Bytes: 2
States: 4

Flags Affected					
Z	N	C	V	VT	ST
1	0	0	0	—	—

14. CLRC — CLEAR CARRY FLAG

Operation: The value of the carry flag is set to zero.
C ← 0

Assembly Language Format: CLRC

Object Code Format: [11111000]

Bytes: 1
States: 4

Flags Affected					
Z	N	C	V	VT	ST
—	—	0	—	—	—

15. CLRVT — CLEAR OVERFLOW TRAP

Operation: The value of the overflow-trap flag is set to zero.

VT ← 0

Assembly Language Format: CLRVT

Object Code Format: [11111100]

Bytes: 1
States: 4

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	0	—

16. CMP — COMPARE WORDS

Operation: The source (rightmost) word operand is subtracted from the destination (leftmost) word operand. The flags are altered but the operands remain unaffected. The carry flag is set as complement of borrow.

(DEST) — (SRC)

Assembly Language Format:

	DST	SRC
CMP	wreg,	waop

Object Code Format: [100010aa] [waop] [wreg]

Bytes: 2 + BEA
States: 4 + CEA

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	—

17. CMPB — COMPARE BYTES

Operation: The source (rightmost) byte operand is subtracted from the destination (leftmost) byte operand. The flags are altered but the operands remain unaffected. The carry flag is set as complement of borrow.

(DEST) — (SRC)

Assembly Language Format:

	DST	SRC
CMPB	breg,	baop

Object Code Format: [100110aa] [baop] [breg]

Bytes: 2 + BEA
States: 4 + CEA

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	—

18. DEC — DECREMENT WORD

Operation: The value of the word operand is decremented by one.

(DEST) ← (DEST) — 1

Assembly Language Format: DEC wreg

Object Code Format: [00000101] [wreg]

Bytes: 2
States: 4

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	—

19. DECB — DECREMENT BYTE

Operation: The value of the byte operand is decremented by one.
(DEST) ← (DEST) — 1

Assembly Language Format: DECB breg

Object Code Format: [00010101] [breg]

Bytes: 2
States: 4

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	—

20. DI — DISABLE INTERRUPTS

Operation: Interrupts are disabled. Interrupt-calls will not occur after this instruction.
Interrupt Enable (PSW.9) ← 0

Assembly Language Format: DI

Object Code Format: [11111010]

Bytes: 1
States: 4

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

21. DIV — DIVIDE INTEGERS

Operation: This instruction divides the contents of the destination LONG-INTEGER operand by the contents of the INTEGER word operand, using signed arithmetic. The low order word of the destination (i.e., the word with the lower address) will contain the quotient; the high order word will contain the remainder.

(low word DEST) ← (DEST) / (SRC)
(high word DEST) ← (DEST) MOD (SRC)
The above two statements are performed concurrently.

Assembly Language Format:

	DST	SRC
DIV	lreg,	waop

Object Code Format: [11111110] [100011aa] [waop] [lreq]

Bytes: 2 + BEA
States: 29 + CEA

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	?	↑	—

22. DIVB — DIVIDE SHORT-INTEGERS

Operation: This instruction divides the contents of the destination INTEGER operand by the contents of the source SHORT-INTEGER operand, using signed arithmetic. The low order byte of the destination (i.e. the byte with the lower address) will contain the quotient; the high order byte will contain the remainder.

(low byte DEST) ← (DEST) / (SRC)
(high byte DEST) ← (DEST) MOD (SRC)
The above two statements are performed concurrently.

Assembly Language Format:

	DST	SRC
DIVB	wreg,	baop

Object Code Format: [11111110] [100111aa] [baop] [wreg]

Bytes: 2 + BEA
States: 21 + CEA

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	?	↑	—

23. DIVU — DIVIDE WORDS

Operation: This instruction divides the content of the destination DOUBLE-WORD operand by the contents of the source WORD operand, using unsigned arithmetic. The low order word will contain the quotient; the high order WORD will contain the remainder.

(low word DEST) ← (DEST) / (SRC)
(high word DEST) ← (DEST) MOD (SRC)
The above two statements are performed concurrently.

Assembly Language Format:

	DST	SRC
DIVU	lreg,	waop

Object Code Format: [100011aa] [waop] [lreq]

Bytes: 2 + BEA
States: 25 + CEA

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	↕	↑	—

24. DIVUB — DIVIDE BYTES

Operation: This instruction divides the contents of the destination WORD operand by the contents of the source BYTE operand, using unsigned arithmetic. The low order byte of the destination, (i.e., the byte with the lower address) will contain the quotient; the high order byte will contain the remainder.

(low byte DEST) ← (DEST) / (SRC)
(high byte DEST) ← (DEST) MOD (SRC)
The above two statements are performed concurrently.

Assembly Language Format:

	DST	SRC
DIVUB	wreg,	baop

Object Code Format: [100111aa] [baop] [wreg]

Bytes: 2 + BEA
States: 17 + CEA

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	↕	↑	—

25. DJNZ — DECREMENT AND JUMP IF NOT ZERO

Operation: The value of the byte operand is decremented by 1. If the result is not equal to 0, the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the result of the decrement is zero then control passes to the next sequential instruction.

```
(COUNT) ← (COUNT) — 1
if (COUNT) <> 0 then
  PC ← PC + disp (sign-extended to 16 bits)
end_if
```

Assembly Language Format: DJNZ breg,cadd

Object Code Format: [11100000] [breg] [disp]

Bytes: 3
States: Jump Not Taken: 5
Jump Taken: 9

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

26. EI — ENABLE INTERRUPTS

Operation: Interrupts are enabled following the execution of the next statement. Interrupt-calls cannot occur immediately following this instruction.

```
Interrupt Enable (PSW.9) ← 1
```

Assembly Language Format: EI

Object Code Format: [11111011]

Bytes: 1
States: 4

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

27. EXT — SIGN EXTEND INTEGER INTO LONG-INTEGER

Operation: The low order word of the operand is sign-extended throughout the high order word of the operand.

```
if (low word DEST) < 8000H then
  (high word DEST) ← 0
else
  (high word DEST) ← 0FFFFH
end_if
```

Assembly Language Format: EXT lreg

Object Code Format: [00000110] [lreg]

Bytes: 2
States: 4

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	0	0	—	—

28. EXTB — SIGN EXTEND SHORT-INTEGER INTO INTEGER

Operation: The low order byte of the operand is sign-extended throughout the high order byte of the operand.

```
if (low byte DEST) < 80H then
  (high byte DEST) ← 0
else
  (high byte DEST) ← 0FFH
end_if
```

Assembly Language Format: EXTB wreg

Object Code Format: [00010110] [wreg]

Bytes: 2
States: 4

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	0	0	—	—

29. INC — INCREMENT WORD

Operation: The value of the word operand is incremented by 1.

(DEST) ← (DEST) + 1

Assembly Language Format: INC wreg

Object Code Format: [00000111] [wreg]

Bytes: 2
States: 4

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	—

30. INCB — INCREMENT BYTE

Operation: The value of the byte operand is incremented by 1.

(DEST) ← (DEST) + 1

Assembly Language Format: INCB breg

Object Code Format: [00010111] [breg]

Bytes: 2
States: 4

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	—

31. JBC — JUMP IF BIT CLEAR

Operation: The specified bit is tested. If it is clear (i.e., 0), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the bit is set (i.e., 1), control passes to the next sequential instruction.

```
if (specified bit) = 0 then
    PC ← PC + disp (sign-extened to 16 bits)
```

Assembly Language Format: JBC breg,bitno,cadd

Object Code Format: [00110bbb] [breg] [disp]
where bbb is the bit number within the specified register.

Bytes: 3
States: Jump Not Taken: 5
Jump Taken: 9

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

32. JBS — JUMP IF BIT SET

Operation: The specified bit is tested. If it is set (i.e., 1), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the bit is clear (i.e., 0), control passes to the next sequential instruction.

```
if (specified bit) = 1 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JBS breg,bitno,cadd

Object Code Format: [00111bbb][breg] [disp]
where bbb is the bit number within the specified register.

Bytes: 3
States: Jump Not Taken: 5
Jump Taken: 9

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

33. JC — JUMP IF CARRY FLAG IS SET

Operation: If the carry flag is set (i.e., 1), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the carry flag is clear (i.e., 0), control passes to the next sequential instruction.

```
if C = 1 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JC cadd

Object Code Format: [11011011] [disp]

Bytes: 2
States: Jump Not Taken: 4
Jump Taken: 8

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

34. JE — JUMP IF EQUAL

Operation: If the zero flag is set (i.e., 1), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the zero flag is clear (i.e., 0), control passes to the next sequential instruction.

```
if Z = 1 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JE cadd

Object Code Format: [11011111] [disp]

Bytes: 2
States: Jump Not Taken: 4
Jump Taken: 8

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

35. JGE — JUMP IF SIGNED GREATER THAN OR EQUAL

Operation: If the negative flag is clear (i.e., 0), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the negative flag is set (i.e., 1), control passes to the next sequential instruction.

```
if N = 1 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JGE cadd

Object Code Format: [11010110] [disp]

Bytes: 2
States: Jump Not Taken: 4
Jump Taken: 8

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

36. JGT — JUMP IF SIGNED GREATER THAN

Operation: If both the negative flag and the zero flag are clear (i.e., 0), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If either the negative flag or the zero flag are set (i.e., 1,) control passes to the next sequential instruction.

```
if N = 0 AND Z = 0 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JGT cadd

Object Code Format: [11010010] [disp]

Bytes: 2
States: Jump Not Taken: 4
Jump Taken: 8

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

37. JH — JUMP IF HIGHER (UNSIGNED)

Operation: If the carry flag is set (i.e., 1), but the zero flag is not, the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If either the carry flag is clear or the zero flag is set, control passes to the next sequential instruction.

```
if C = 1 AND Z = 0 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JH cadd

Object Code Format: [11011001] [disp]

Bytes: 2
States: Jump Not Taken: 4
Jump Taken: 8

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

38. JLE — JUMP IF SIGNED LESS THAN OR EQUAL

Operation: If either the negative flag or the zero flag are set (i.e., 1), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to + 127. If both the negative flag and the zero flag are clear (i.e., 0), control passes to the next sequential instruction.

```
if N = 1 OR Z = 1 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JLE cadd

Object Code Format: [11011010] [disp]

Bytes: 2
States: Jump Not Taken: 4
Jump Taken: 8

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

39. JLT — JUMP IF SIGNED LESS THAN

Operation: If the negative flag is set (i.e., 1), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the negative flag is clear (i.e., 0), control passes to the next sequential instruction.

```
if N = 1 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JLT cadd

Object Code Format: [11011110] [disp]

Bytes: 2
States: Jump Not Taken: 4
Jump Taken: 8

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

40. JNC — JUMP IF CARRY FLAG IS CLEAR

Operation: If the carry flag is clear (i.e., 0), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the carry flag is set (i.e., 1), control passes to the next sequential instruction.

```
if C = 0 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JNC cadd

Object Code Format: [11010011] [disp]

Bytes: 2
States: Jump Not Taken: 4
Jump Taken: 8

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

41. JNE — JUMP IF NOT EQUAL

Operation: If the zero flag is clear (i.e., 0), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the zero flag is set (i.e., 1), control passes to the next sequential instruction.

```
if Z = 0 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JNE cadd

Object Code Format: [11010111 [disp]

Bytes: 2
States: Jump Not Taken: 4
Jump Taken: 8

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

42. JNH — JUMP IF NOT HIGHER (UNSIGNED)

Operation: If either the carry flag is clear (i.e., 0), or the zero flag is set (i.e., 1), the distance from the end of this instruction to the target label is added to program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to + 127. If the carry flag is set (i.e., 1) and the zero flag is not, control passes to the next sequential instruction.

```
if C = 0 OR Z = 1 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JNH cadd

Object Code Format: [11010001] [disp]

Bytes: 2
States: Jump Not Taken: 4
Jump Taken: 8

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

43. JNST — JUMP IF STICKY BIT IS CLEAR

Operation: If the sticky bit flag is clear (i.e., 0), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the sticky bit flag is set (i.e., 1), control passes to the next sequential instruction.

```
if ST = 0 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JNST cadd

Object Code Format: [11010000] [disp]

Bytes: 2
States: Jump Not Taken: 4
Jump Taken: 8

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

44. JNV — JUMP IF OVERFLOW FLAG IS CLEAR

Operation: If the overflow flag is clear (i.e., 0), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the overflow flag is set (i.e., 1), control passes to next sequential instruction.

```
if V = 0 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JNV cadd

Object Code Format: [11010101] [disp]

Bytes: 2
States: Jump Not Taken: 4
Jump Taken: 8

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

45. JNVT — JUMP IF OVERFLOW TRAP IS CLEAR

Operation: If the overflow trap flag is clear (i.e., 0), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the overflow trap flag is set (i.e., 1), control passes to the next sequential instruction. The VT flag is cleared.

```
if VT = 0 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JNVT cadd

Object Code Format: [11010100] [disp]

Bytes: 2
States: Jump Not Taken: 4
Jumps Taken: 8

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	0	—

46. JST — JUMP IF STICKY BIT IS SET

Operation: If the sticky bit flag is set (i.e., 1), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the sticky bit flag is clear (i.e., 0), control passes to the next sequential instruction.

```
if ST = 1 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JST cadd

Object Code Format: [11011000] [disp]

Bytes: 2
State: Jump Not Taken: 4
Jump Taken: 8

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

47. JV — JUMP IF OVERFLOW FLAG IS SET

Operation: If the overflow is set (i.e., 1), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the overflow flag is clear (i.e., 0), control passes to the next sequential instruction.

```
if V = 1 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JV cadd

Object Code Format: [11011101] [disp]

Bytes: 2
States: Jump Not Taken: 4
Jump Taken: 8

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

48. JVT — JUMP IF OVERFLOW TRAP IS SET

Operation: If the overflow trap flag is set (i.e., 1), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the overflow trap flag is clear (i.e., 0), control passes to the next sequential instruction. The VT flag is cleared.

```
if VT = 1 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JVT cadd

Object Code Format: [11011100] [disp]

Bytes: 2
States: Jump Not Taken: 4
Jump Taken: 8

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	0	—

49. LCALL — LONG CALL

Operation: The contents of the program counter (the return address) is pushed onto the stack. Then the distance from the end of this instruction to the target label is added to the program counter, effecting the call. The operand may be any address in the entire address space.

```
SP ← SP − 2
(SP) ← PC
PC ← PC + disp
```

Assembly Language Format: LCALL cadd

Object Code Format: [11101111] [disp-low] [disp-hi]

Bytes: 3
States: Onchip stack: 13
Offchip stack: 16

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

50. LD — LOAD WORD

Operation: The value of the source (rightmost) word operand is stored into the destination (leftmost) operand.

(DEST) ← (SRC)

Assembly Language Format:

```
      DST    SRC
LD    wreg,  waop
```

Object Code Format: [101000aa] [waop] [wreg]

Bytes: 2 + BEA
States: 4 + CEA

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

51. LDB — LOAD BYTE

Operation: The value of the source (rightmost) byte operand is stored into the destination (leftmost) operand.

(DEST) ← (SRC)

Assembly Language Format:

```
      DST    SRC
LDB   breg,  baop
```

Object Code Format: [101100aa] [baop] [breg]

Bytes: 2 + BEA
States: 4 + CEA

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

52. LDBSE — LOAD INTEGER WITH SHORT-INTEGER

Operation: The value of the source (rightmost) byte operand is sign-extended and stored into the destination (leftmost) word operand.

```
(low byte DEST) ← (SRC)
if (SRC) < 80H then
    (high byte DEST) ← 0
else
    (high byte DEST) ← 0FFH
end_if
```

Assembly Language Format:

```
          DST     SRC
LDBSE     wreg,   baop
```

Object Code Format: [101111aa] [baop] [wreg]

Bytes: 2 + BEA
States: 4 + CEA

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

53. LDBZE — LOAD WORD WITH BYTE

Operation: The value of the source (rightmost) byte operand is zero-extended and stored into the destination (leftmost) word operand.

```
(low byte DEST) ← (SRC)
(high byte DEST) ← 0
```

Assembly Language Format:

```
          DST     SRC
LDBZE     wreg,   baop
```

Object Code Format: [101011aa] [baop] [wreg]

Bytes: 2 + BEA
States: 4 + CEA

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

54. LJMP — LONG JUMP

Operation: The distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The operand may be any address in the entire address space.

PC ← PC + disp

Assembly Language Format: LJMP cadd

Object Code Format: [11100111] [disp-low] [disp-hi]

Bytes: 3
States: 8

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

55. MUL (Two Operands) — MULTIPLY INTEGERS

Operation: The two INTEGER operands are multiplied using signed arithmetic and the 32-bit result is stored into the destination (leftmost) LONG-INTEGER operand. The sticky bit flag is undefined after the instruction is executed.

(DEST) ← (DEST) * (SRC)

Assembly Language Format:

	DST	SRC
MUL	lreg,	waop

Object Code Format: [11111110] [011011aa] [waop] [lreg]

Bytes: 3 + BEA
States 29 + CEA

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	?

56. MUL (Three Operands) — MULTIPLY INTEGERS

Operation: The second and third INTEGER operands are multiplied using signed arithmetic and the 32-bit result is stored into the destination (leftmost) LONG INTEGER operand. The sticky bit flag is undefined after the instruction is executed.

(DEST) ← (SRC1) * (SRC2)

Assembly Language Format:

	DST	SRC1	SRC2
MUL	lreg,	wreg,	waop

Object Code Format: [11111110] [010011aa] [waop] [wreg] [lreg]

Bytes: 4 + BEA
States: 30 + CEA

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	?

57. MULB (Two Operands) — MULTIPLY SHORT-INTEGERS

Operation: The two SHORT-INTEGER operands are multiplied using signed arithmetic and the 16-bit result is stored into the destination (leftmost) INTEGER operand. The sticky bit flag is undefined after the instruction is executed.

(DEST) ← (DEST) * (SRC)

Assembly Language Format:

	DST	SRC
MULB	wreg,	baop

Object Code Format: [11111110] [011111aa] [baop] [wreg]

Bytes: 3 + BEA
States: 21 + CEA

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	?

58. MULB (Three Operands) — MULTIPLY SHORT-INTEGERS

Operation: The second and third SHORT-INTEGER operands are multiplied using signed arithmetic and the 16-bit result is stored into the destination (leftmost) INTEGER operand. The sticky bit flag is undefined after the instruction is executed.

(DEST) ← (SRC1) * (SRC2)

Assembly Language Format:

	DST	SRC1	SRC2
MULB	wreg,	breg	baop

Object Code Format: [11111110] [010111aa] [baop] [breg] [wreg]

Bytes: 4 + BEA
States: 22 + CEA

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	?

59. MULU (Two Operands) — MULTIPLY WORDS

Operation: The two WORD operands are multiplied using unsigned arithmetic and the 32-bit result is stored into the destination (leftmost) DOUBLE-WORD operand. The sticky bit flag is undefined after the instruction is executed.

(DEST) ← (DEST) * (SRC)

Assembly Language Format:

	DST	SRC
MULU	lreg,	waop

Object Code Format: [011011aa] [waop] [lreg]

Bytes: 2 + BEA
States: 25 + CEA

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	?

60. MULU (Three Operands) — MULTIPLY WORDS

Operation: The second and third WORD operands are multiplied using unsigned arithmetic and the 32-bit result is stored into the destination (leftmost) DOUBLEWORD operand. The sticky bit flag is undefined after the instruction is executed.

(DEST) ← (SRC1) * (SRC2)

Assembly Language Format:

```
        DST     SRC1    SRC2
MULU    lreg,   wreg,   waop
```

Object Code Format: [010011aa] [waop] [wreg] [lreg]

Bytes: 3 + BEA
States: 26 + CEA

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	?

61. MULUB (Two Operands) — MULTIPLY BYTES

Operation: The two BYTE operands are multiplied using unsigned arithmetic and the WORD result is stored into the destination (leftmost) operand. The sticky bit flag is undefined after the instruction is executed.

(DEST) ← (DEST) * (SRC)

Assembly Language Format:

```
        DST     SRC
MULUB   wreg,   baop
```

Object Code Format: [011111aa] [baop] [wreg]

Bytes: 2 + BEA
States: 17 + CEA

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	?

62. MULUB (Three Operands) — MULTIPLY BYTES

Operation: The second and third BYTE operands are multiplied using unsigned arithmetic and the WORD result is stored into the destination (leftmost) operand. The sticky bit flag is undefined after the instruction is executed.

(DEST) ← (SRC1) * (SRC2)

Assembly Language Format:

	DST	SRC1	SRC2
MULUB	wreg,	breg,	baop

Object Code Format: [010111aa] [baop] [breg] [wreg]

Bytes: 3 + BEA
States: 18 + CEA

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	?

63. NEG — NEGATE INTEGER

Operation: The value of the INTEGER operand is negated.

(DEST) ← −(DEST)

Assembly Language Format: NEG wreg

Object Code Format: [00000011] [wreg]

Bytes: 2
States: 4

Flags Affected					
Z	N	C	V	VT	ST
↙	↙	↙	↙	↑	—

64. NEGB — NEGATE SHORT-INTEGER

Operation: The value of the SHORT-INTEGER operand is negated.
(DEST) ← −(DEST)

Assembly Language Format: NEGB breg

Object Code Format: [00010011] [breg]

Bytes: 2
States: 4

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	—

65. NOP — NO OPERATION

Operation: Nothing is done. Control passes to the next sequential instruction.

Assembly Language Format: NOP

Object Code Format: [11111101]

Bytes: 1
States: 4

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

66. NORML — NORMALIZE LONG-INTEGER

Operation: The LONG-INTEGER operand is normalized; i.e., it is shifted to the left until its most significant bit is 1. If the most significant bit is still 0 after 31 shifts, the process stops and the zero flag is set. The number of shifts actually performed is stored in the second operand.

```
(COUNT) ← 0
do while (MSB(DEST) = 0) AND ((COUNT) < 31)
    (DEST) ← (DEST) * 2
    (COUNT) ← (COUNT) + 1
end_while
```

Assembly Language Format: NORML lreg,breg

Object Code Format: [00001111] [breg] [lreg]

Bytes: 3
States: 8 + No. of shifts performed

Flags Affected					
Z	N	C	V	VT	ST
✓	?	0	—	—	—

67. NOT — COMPLEMENT WORD

Operation: The value of the WORD operand is complemented: each 1 is replaced with a 0, and each 0 with a 1.

```
(DEST) ← NOT (DEST)
```

Assembly Language Format: NOT wreg

Object Code Format: [00000010] [wreg]

Bytes 2
States: 4

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	0	0	—	—

68. NOTB — COMPLEMENT BYTE

Operation: The vaule of the BYTE operand is complemented: each 1 is replaced with a 0, and each 0 with a 1.

(DEST) ← NOT (DEST)

Assembly Language Format: NOTB breg

Object Code Format: [00010010] [breg]

Bytes: 2
States: 4

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	0	0	—	—

69. OR — LOGICAL OR WORDS

Operation: The source (rightmost) WORD is ORed with the destination (leftmost) WORD operand. Each bit is set to 1 if the corresponding bit in either the source operand or the destination operand is 1. The result replaces the original destination operand.

(DEST) ← (DEST) OR (SRC)

Assembly Language Format:

	DST	SRC
OR	wreg,	waop

Object Code Format: [100000aa] [waop] [wreg]

Bytes: 2 + BEA
States: 4 + CEA

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	0	0	—	—

70. ORB — LOGICAL OR BYTES

Operation: The source (rightmost) BYTE operand is ORed with the destination (leftmost) BYTE operand. Each bit is set to 1 if the corresponding bit in either the source operand or the destination operand was 1. The result replaces the original destination operand.

(DEST) ← (DEST) OR (SRC)

Assembly Language Format: ORB breg,baop

Object Code Format: [100100aa] [baop] [breg]

Bytes: 2 + BEA
States: 4 + CEA

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	0	0	—	—

71. POP — POP WORD

Operation: The word on top of the stack is popped and placed at the destination operand.

(DEST) ← (SP)
SP ← SP + 2

Assembly Language Format: POP waop

Object Code Format: [110011aa] [waop]

Bytes 1 + BEA
States: Onchip Stack: 12 + CEA
Offchip Stack: 14 + CEA

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

72. POPF — POP FLAGS

Operation: The word on top of the stack is popped and placed in the PSW. Interrupt calls cannot occur immediately following this instruction.

(PSW) ← (SP)
SP ← SP + 2

Assembly Language Format: POPF

Object Code Format: [11110011]

Bytes: 1
States: Onchip Stack: 9
Offchip Stack: 13

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	✓	✓

73. PUSH — PUSH WORD

Operation: The specified operand is pushed onto the stack.

SP ← SP − 2
(SP) ← (DEST)

Assembly Language Format: PUSH waop

Object Code Format: [110010aa] [waop]

Bytes: 1 + BEA
States: Onchip Stack: 8 + CEA
Offchip Stack: 12 + CEA

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

74. PUSHF — PUSH FLAGS

Operation: The PSW is pushed on top of the stack, and then set to all zeroes. This implies that all interrupts are disabled. Interrupt-calls cannot occur immediately following this instruction.

```
SP ← SP − 2
(SP) ← PSW
PSW ← 0
```

Assembly Language Format: PUSHF

Object Code Format: [11110010]

Bytes: 1
States: Onchip Stack: 8
Offchip Stack: 12

Flags Affected					
Z	N	C	V	VT	ST
0	0	0	0	0	0

75. RET — RETURN FROM SUBROUTINE

Operation: The PC is popped off the top of the stack.

```
PC ← (SP)
SP ← SP + 2
```

Assembly Language Format: RET

Object Code Format: [11110000]

Bytes: 1
States: Onchip Stack: 12
Offchip Stack: 16

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

76. RST — RESET SYSTEM

Operation: The PSW is initialized to zero, and the PC is initialized to 2080H. The I/O registers are set to their initial value. Executing this instruction will cause a pulse to appear on the reset pin of the 8096.

PSW ← 0
PC ← 2080H

Assembly Language Format: RST

Object Code Format: [11111111]

Bytes: 1
States: 16

Flags Affected					
Z	N	C	V	VT	ST
0	0	0	0	0	0

77. SCALL — SHORT CALL

Operation: The contents of the program counter (the return address) is pushed onto the stack. Then the distance from the end of this instruction to the target label is added to the program counter, effecting the call. The offset from the end of this instruction to the target label must be in the range of −1024 to +1023 inclusive.

SP ← SP − 2
(SP) ← PC
PC ← PC + disp (sign-extended to 16 bits)

Assembly Language Format: SCALL cadd

Object Code Format: [00101xxx] [disp-low]

where xxx holds the three high-order bits of displacement.

Bytes: 2
States Onchip Stack: 13
Offchip Stack: 16

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

78. SETC — SET CARRY FLAG

Operation: The carry flag is set.

```
C ← 1
```

Assembly Language Format: SETC

Object Code Format: [11111001]

Bytes: 1
States: 4

Flags Affected					
Z	N	C	V	VT	ST
—	—	1	—	—	—

79. SHL — SHIFT WORD LEFT

Operation: The destination (leftmost) word operand is shifted left as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 (0FH) inclusive, or as the content of any register. Details on indirect shifts can be found in the Overview. The right bits of the result are filled with zeroes. The last bit shifted out is saved in the carry flag.

```
Temp ← (COUNT)
do while Temp <> 0
    C ← High order bit of (DEST)
    (DEST) ← (DEST) * 2
    Temp ← Temp — 1
end_while
```

Assembly Language Format:

```
        SHL     wreg,#count
or
        SHL     wreg,breg
```

Object Code Format: [00001001] [cnt/breg] [wreg]

Bytes: 3
States: 7 + No. of shifts performed
note: 0 place shifts take 8 states.

Flags Affected					
Z	N	C	V	VT	ST
✓	?	✓	✓	↑	—

80. SHLB — SHIFT BYTE LEFT

Operation: The destination (leftmost) byte operand is shifted left as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 (0FH) inclusive, or as the content of any register. Details on indirect shifts can be found in the Overview. The right bits of the result are filled with zeroes. The last bit shifted out is saved in the carry flag.

```
Temp ← (COUNT)
do while Temp <> 0
    C ← High order bit of (DEST)
    (DEST) ← (DEST) * 2
    TEMP ← Temp − 1
end_while
```

Assembly Language Format:

```
        SHLB    breg,#count
or
        SHLB    breg,breg
```

Object Code Format: [00011001] [cnt/breg] [breg]

Bytes 3
States: 7 + No. of shifts performed
note: 0 place shifts take 8 states.

Flags Affected					
Z	N	C	V	VT	ST
✓	?	✓	✓	↑	—

81. SHLL — SHIFT DOUBLE-WORD LEFT

Operation: The destination (leftmost) double-word operand is shifted left as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 (0FH) inclusive, or as the content of any register. Details on indirect shifts can be found in the Overview. The right bits of the result are filled with zeroes. The last bit shifted out is saved in the carry flag.

```
Temp ← (COUNT)
do while Temp <> 0
    C ← High order bit of (DEST)
    (DEST) ← (DEST) * 2
    Temp ← Temp − 1
end_while
```

Assembly Language Format:

```
        SHLL    lreg,#count
or
        SHLL    lreg, breg
```

Object Code Format: [00001101] [cnt/breg] [lreg]

Bytes: 3
States: 7 + No. of shifts performed
note: 0 place shifts take 8 states.

Flags Affected					
Z	N	C	V	VT	ST
✓	?	✓	✓	↑	—

82. SHR — LOGICAL RIGHT SHIFT WORD

Operation: The destination (leftmost) word operand is shifted right as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 (0FH) inclusive, or as the content of any register. Details on indirect shifts can be found in the Overview. The left bits of the result are filled with zeroes. The last bit shifted out is saved to the carry. The sticky bit flag is cleared at the beginning of the instruction, and set if at any time during the shift a 1 is shifted first into the carry flag, and a further shift cycle occurs.

```
Temp ← (COUNT)
do while Temp <> 0
    C ← Low order bit of (DEST)
    (DEST) ← (DEST) / 2 where / is unsigned division
    Temp ← Temp − 1
end_while
```

Assembly Language Format:

```
        SHR     wreg,#count
or
        SHR     wreg,breg
```

Object Code Format: [00001000] [cnt/breg] [wreg]

Bytes: 3
States: 7 + No. of shifts performed
note: 0 place shifts take 8 states.

Flags Affected					
Z	N	C	V	VT	ST
✓	0	✓	0	—	✓

83. SHRA — ARITHMETIC RIGHT SHIFT WORD

Operation: The destination (leftmost) word operand is shifted right as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 (0FH) inclusive, or as the content of any register. Details on indirect shifts can be found in the Overview. If the original high order bit value was 0, zeroes are shifted in. If the value was 1, ones are shifted in. The last bit shifted out is saved in the carry. The sticky bit flag is cleared at the beginning of the instruction, and set if at any time during the shift a 1 is shifted first into the carry flag, and a further shift cycle occurs.

```
Temp ← (COUNT)
do while Temp <> 0
    C ← Low order bit of (DEST)
    (DEST) ← (DEST) / 2 where / is signed division
    Temp ← Temp − 1
end_while
```

Assembly Language Format:

```
        SHRA    wreg,#count
or
        SHRA    wreg,breg
```

Object Code Format: [00001010] [cnt/breg] [wreg]

Bytes: 3
States: 7 + No. of shifts performed
note: 0 place shifts take 8 states.

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	0	—	✓

84. SHRAB — ARITHMETIC RIGHT SHIFT BYTE

Operation: The destination (leftmost) byte operand is shifted right as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 (0FH) inclusive, or as the content of any register. Details on indirect shifts can be found in the Overview. If the original high order bit value was 0, zeroes are shifted in. If that value was 1, ones are shifted in. The last bit shifted out is saved in the carry. The sticky bit flag is cleared at the beginning of the instruction, and set if at any time during the shift a 1 is shifted first into the carry flag, and a further shift cycle occurs.

```
Temp ← (COUNT)
do while Temp <> 0
    C, = Low order bit of (DEST)
    (DEST) ← (DEST) / 2 where / is signed division
    Temp ← Temp − 1
end_while
```

Assembly Language Format:

```
        SHRAB   breg,#count
or
        SHRAB   breg,breg
```

Object Code Format: [00011010] [cnt/breg] [breg]

Bytes: 3
States: 7 + No. of shifts performed
note: 0 place shifts take 8 states.

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	0	—	✓

85. SHRAL — ARITHMETIC RIGHT SHIFT DOUBLE-WORD

Operation: The destination (leftmost) double-word operand is shifted right as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 (0FH) inclusive, or as the content of any register. Details on indirect shifts can be found in the Overview. If the original high order bit value was 0, zeroes are shifted in. If the value was 1, ones are shifted in. The sticky bit is cleared at the beginning of the instruction, and set if at any time during the shift a 1 is shifted first into the carry flag, and a further shift cycle occurs.

```
Temp ← (COUNT)
do while Temp <> 0
    C ← Low order bit of (DEST)
    (DEST) ← (DEST) / 2 where / is signed division
    Temp ← Temp − 1
end_while
```

Assembly Language Format:

```
        SHRAL   lreg,#count
or
        SHRAL   lreg,breg
```

Object Code Format: [00001110] [cnt/breg] [lreg]

Bytes: 3
States: 7 + No. of shifts performed
note: 0 place shifts take 8 states.

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	0	—	✓

86. SHRB — LOGICAL RIGHT SHIFT BYTE

Operation: The destination (leftmost) byte operand is shifted right as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 (0FH) inclusive, or as the content of any register. Details on indirect shifts can be found in the Overview. The left bits of the result are filled with zeroes. The last bit shifted out is saved in the carry. The sticky bit flag is cleared at the beginning of the instruction, and set if at any time during the shift a 1 is shifted first into the carry flag, and a further shift cycle occurs.

```
Temp ← (COUNT)
do while Temp <> 0
    C ← Low order bit of (DEST)
    (DEST) ← (DEST) / 2 where / is unsigned division
    Temp ← Temp − 1
end_while
```

Assembly Language Format:

```
        SHRB    breg,#count
or
        SHRB    breg,breg
```

Object Code Format: [00011000] [cnt/breg] [breg]

Bytes: 3
States: 7 + No. of shifts performed
note: 0 place shifts take 8 states.

Flags Affected					
Z	N	C	V	VT	ST
✓	0	✓	0	—	✓

87. SHRL — LOGICAL RIGHT SHIFT DOUBLE-WORD

Operation: The destination (leftmost) double-word operand is shifted right as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 (0FH) inclusive, or as the content of any register. Details on indirect shifts can be found in the Overview. The left bits of the result are filled with zeroes. The last bit shifted out is saved in the carry. The sticky bit flag is cleared at the beginning of the instruction, and set if at any time during the shift a 1 is shifted first into the carry flag, and a further shift cycle occurs.

```
Temp ← (COUNT)
do while Temp <> 0
    C ← Low order bit of (DEST)
    (DSET) ← (DEST) / 2 where / is unsigned division
    Temp ← Temp − 1
end_while
```

Assembly Language Format:

```
        SHRL    lreg,#count
or
        SHRL    lreg,breg
```

Object Code Format: [00001100] [cnt/breg] [lreg]

Bytes: 3
States: 7 + No. of shifts performed
note: 0 place shifts take 8 states.

Flags Affected					
Z	N	C	V	VT	ST
✓	0	✓	0	—	✓

88. SJMP — SHORT JUMP

Operation: The distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the label must be in the range of −1024 to +1023 inclusive.

PC ← PC + disp (sign-extended to 16 bits)

Assembly Language Format: SJMP cadd

Object Code Format: [00100xxx] [disp-low]
where xxx holds the three high order bits of the displacement.

Bytes: 2
States: 8

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

89. SKIP — TWO BYTE NO-OPERATION

Operation: Nothing is done. This is actually a two-byte NOP where the second byte can be any value, and is simply ignored. Control passes to the next sequential instruction.

Assembly Language Format: SKIP breg

Object Code Format: [00000000] [breg]

Bytes: 2
States: 4

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

90. ST — STORE WORD

Operation: The value of the leftmost word operand is stored into the rightmost operand.
(DEST) ← (SRC)

Assembly Language Format:

	SRC	DST
ST	wreg,	waop

Object Code Format: [110000aa] [waop] [wreg]

Bytes: 2 + BEA
States: 4 + CEA

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

91. STB — STORE BYTE

Operation: The value of the leftmost byte operand is stored into the rightmost operand.
(DEST) ← (SRC)

Assembly Language Format:

	SRC	DST
STB	breg,	baop

Object Code Format: [110001aa] [baop] [breg]

Bytes: 2 + BEA
States: 4 + CEA

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

92. SUB (Two Operands) — SUBTRACT WORDS

Operation: The source (rightmost) word operand is subtracted from the destination (leftmost) word operand, and the result is stored in the destination. The carry flag is set as complement of borrow.

(DEST) ← (DEST) — (SRC)

Assembly Language Format:

```
        DST     SRC
SUB     wreg,   waop
```

Object Code Format: [011010aa] [waop] [wreg]

Bytes: 2 + BEA
States: 4 + CEA

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	—

93. SUB (Three Operands) — SUBTRACT WORDS

Operation: The source (rightmost) word operand is subtracted from the second word operand, and the result is stored in the destination (the leftmost operand). The carry flag is set as complement of borrow.

(DEST) ← (SRC1) — (SRC2)

Assembly Language Format:

```
        DST     SRC1    SRC2,
SUB     wreg,   wreg,   waop
```

Object Code Format: [010010aa] [waop] [Sweg] [Dwreg]

Bytes: 3 + BEA
States: 5 + CEA

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	—

94. SUBB (Two Operands) — SUBTRACT BYTES

Operation: The source (rightmost) byte is subtracted from the destination (leftmost) byte operand, and the result is stored in the destination. The carry flag is set as complement of borrow.

(DEST) ← (DEST) — (SRC)

Assembly Language Format:

```
        DST     SRC
SUBB    breg,   baop
```

Object Code Format: [011110aa] [baop] [breg]

Bytes: 2 + BEA
States 4 + CEA

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	—

95. SUBB (Three Operands) — SUBTRACT BYTES

Operation: The source (rightmost) byte operand is subtracted from the second byte operand, and the result is stored in the destination (the leftmost operand). The carry flag is set as complement of borrow.

(DEST) ← (SRC1) — (SRC2)

Assembly Language Format:

```
        DST     SRC1    SRC2
SUBB    breg,   Sbreg   baop
```

Object Code Format: [010110aa] [baop] [Sbreg] [Dbreg]

Bytes: 3 + BEA
States: 5 + CEA

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	—

96. SUBC — SUBTRACT WORDS WITH BORROW

Operation: The source (rightmost) word operand is subtracted from the destination (leftmost) word operand. If the carry flag was clear, 1 is substracted from the above result. The result replaces the orignal destination operand. The carry flag is set as complement of borrow.

(DEST) ← (DEST) — (SRC) — (1-C)

Assembly Language Format:

	DST	SRC
SUBC	wreg,	waop

Object Code Format: [101010aa] [waop] [wreg]

Bytes: 2 + BEA
States: 4 + CEA

Flags Affected					
Z	N	C	V	VT	ST
↓	✓	✓	✓	↑	—

97. SUBCB — SUBTRACT BYTES WITH BORROW

Operation: The source (rightmost) byte operand is subtracted from the destination (leftmost) byte operand. If the carry flag was clear, 1 is subtracted from the above result. The result replaces the original destination operand. The carry flag is set as complement of borrow.

(DEST) ← (DEST) — (SRC) — (1-C)

Assembly Language Format:

	DST	SRC
SUBCB	breg,	baop

Object Code Format: [101110aa] [baop] [breg]

Bytes: 2 + BEA
States 4 + CEA

Flags Affected					
Z	N	C	V	VT	ST
↓	✓	✓	✓	↑	—

98. TRAP — SOFTWARE TRAP

Operation: This instruction causes an interrupt-call which is vectored through location 2010H. The operation of this instruction is not effected by the state of the interrupt enable flag in the PSW (I). Interrupt-calls cannot occur immediately following this instruction. This instruction is intended for use by Intel provided development tools. These tools will not support user-application of this instruction.

```
SP ← SP − 2
(SP) ← PC
PC ← (2010H)
```

Assembly Language Format: This instruction is not supported by revision 1.0 of the 8096 assembly language.

Object Code Format: [11110111]

Bytes:		1
States	Onchip Stack:	21
	Offchip Stack:	24

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

99. XOR — LOGICAL EXCLUSIVE-OR WORDS

Operation: The source (rightmost) word operand is XORed with the destination (leftmost) word operand. Each bit is set to 1 if the corresponding bit in either the source operand or the destination operand was 1, but not both. The result replaces the original destination operand.

```
(DEST) ← (DEST) XOR (SRC)
```

Assembly Language Format:

```
        DST     SRC
XOR     wreg,   waop
```

Object Code Format: [100001aa] [waop] [wreg]

Bytes:	2 + BEA
States:	4 + CEA

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	0	0	—	—

100. XORB — LOGICAL EXCLUSIVE-OR BYTES

Operation: The source (rightmost) byte operand is XORed with the destination (leftmost) byte operand. Each bit is set to 1 if the corresponding bit in either the source operand or the destination operand was 1, but not both. The result replaces the original destination operand.

(DEST) ← (DEST) XOR (SRC)

Assembly Language Format:

	DST	SRC
XORB	breg,	baop

Object Code Format: [100101aa] [baop] [breg]

Bytes: 2 + BEA
States: 4 + CEA

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	0	0	—	—

MCS®-96 Hardware Design Information

14

MCS®-96 HARDWARE DESIGN INFORMATION

OVERVIEW

This Chapter of the manual is devoted to the hardware engineer. All of the information you need to connect the correct pin to the correct external circuit is provided. Many of the special function pins have different characteristics which are under software control, therefore, it is necessary to define the system completely before the hardware is wired-up.

Frequently within this chapter a specification for a current, voltage, or time period is referred to; the values provided are to be used as an approximation only. The exact specification can be found in the latest data sheet for the particular part and temperature range that is being used.

1.0 REQUIRED HARDWARE CONNECTIONS

Although the 8096BH is a single-chip microcontroller, it still requires several external connections to make it work. Power must be applied, a clock source provided, and some form of reset circuitry must be present. We will look at each of these areas of circuitry separately. Figure 6 shows the connections that are needed for a single-chip system.

1.1 Power Supply Information

Power for the 8096BH flows through six pins; one VCC pin, two VSS pins, one VREF (analog VCC), one ANGND (Analog VSS), and one VPD (V Power Down) pin. All six of these pins **must** be connected on the 8096BH for normal operation. The VCC pin, VREF pin and VPD pin should be tied to 5 volts. The two VSS pins and the ANGND pin must be grounded. When the analog to digital converter is being used it may be desirable to connect the VREF pin to a separate power supply, or at least a separate power supply line.

The two VSS pins should be connected together with as short a lead as possible to avoid problems due to voltage drops across the wiring. There should be no measurable voltage difference between VSS1 and VSS2. The two VSS pins and the ANGND pin must all be nominally at 0 volts. The maximum current drain of the 8096BH is around 180 mA, with all lines unloaded.

When the analog converter is being used, clean, stable power must be provided to the analog section of the chip to assure highest accuracy. To achieve this, it may be desirable to separate the analog power supply from the digital power supply. The VREF pin supplies the digital circuitry in the A/D converter and provides the 5 volt reference to the analog portion of the converter. Similarly, the ANGND pin is ground for all A/D converter circuitry. VREF and ANGND must be connected even if the A/D converter is not used. More information on the analog power supply is in Section 3.1.

1.2 Other Needed Connections

Several other connections are needed to configure the 8096BH. In normal operation the following pins should be connected to the indicated power supply.

Pin		Power Supply
8X9XBH	**8X9X**	
NMI	NMI	VCC
	$\overline{\text{TEST}}$	VCC
$\overline{\text{EA}}$	$\overline{\text{EA}}$	VCC (to allow internal execution)
		VSS (to force external execution)

Although the $\overline{\text{EA}}$ pin has an internal pulldown, it is best to tie this pin to the desired level if it is not left completely disconnected. This will prevent induced noise from disturbing the system. With the exception of 8X9X devices, raising $\overline{\text{EA}}$ to +12.75 volts will place an 8096BH in a special operating mode designed for programming and program memory verification (see Section 10).

1.3 Oscillator Information

The 8096BH requires a clock source to operate. This clock can be provided to the chip through the XTAL1 input or the on-chip oscillator can be used. The frequency of operation is from 6 MHz to 12 MHz.

The on-chip circuitry for the 8096BH oscillator is a single stage linear inverter as shown in Figure 1. It is intended for use as a crystal-controlled, positive reactance oscillator with external connections as shown in Figure 2. In this application, the crystal is being operated in its fundamental response mode as an inductive

reactance in parallel resonance with capacitance external to the crystal.

The crystal specifications and capacitance values (C1 and C2 in Figure 2) are not critical. Thirty pF can be used in these positions at any frequency with good quality crystals. For 0.5% frequency accuracy, the crystal frequency can be specified at series resonance or for parallel resonance with any load capacitance. (In other words, for that degree of frequency accuracy, the load capacitance simply doesn't matter.) For 0.05% frequency accuracy the crystal frequency should be specified for parallel resonance with 25 pF load capacitance, if C1 and C2 are 30 pF.

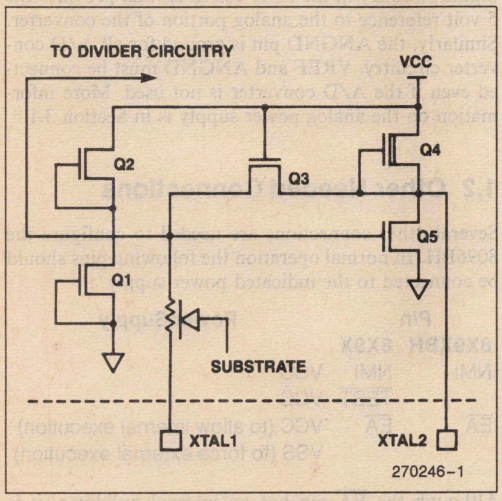


Figure 1. 8096BH Oscillator Circuit

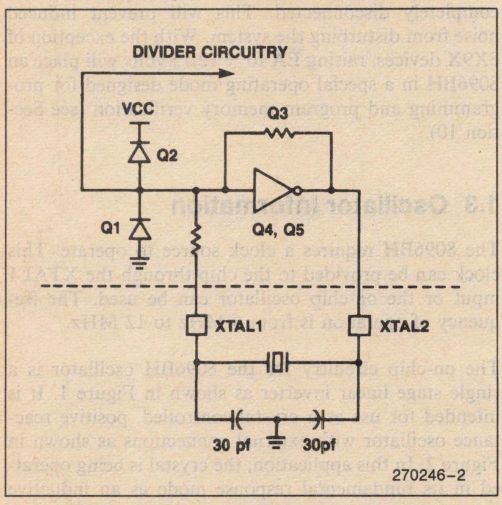


Figure 2. Crystal Oscillator Circuit

A more in-depth discussion of crystal specifications and the selection of values for C1 and C2 can be found in the Intel Application Note, AP-155, "Oscillators for Microcontrollers."

To drive the 8096BH with an external clock source, apply the external clock signal to XTAL1 and let XTAL2 float. An example of this circuit is shown in Figure 3. The required voltage levels on XTAL1 are specified in the data sheet. The signal on XTAL1 must be clean with good solid levels.

It is important that the minimum high and low times are met to avoid having the XTAL1 pin in the transition range for long periods of time. The longer the signal is in the transition region, the higher the probability that an external noise glitch could be seen by the clock generator circuitry. Noise glitches on the 8096BH internal clock lines will cause unreliable operation.

The clock generator provides a 3 phase clock output from the XTAL1 pin input. Figure 4 shows the waveforms of the major internal timing signals.

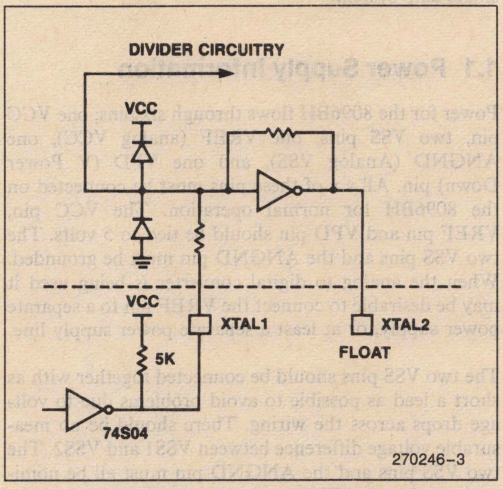


Figure 3. External Clock Drive

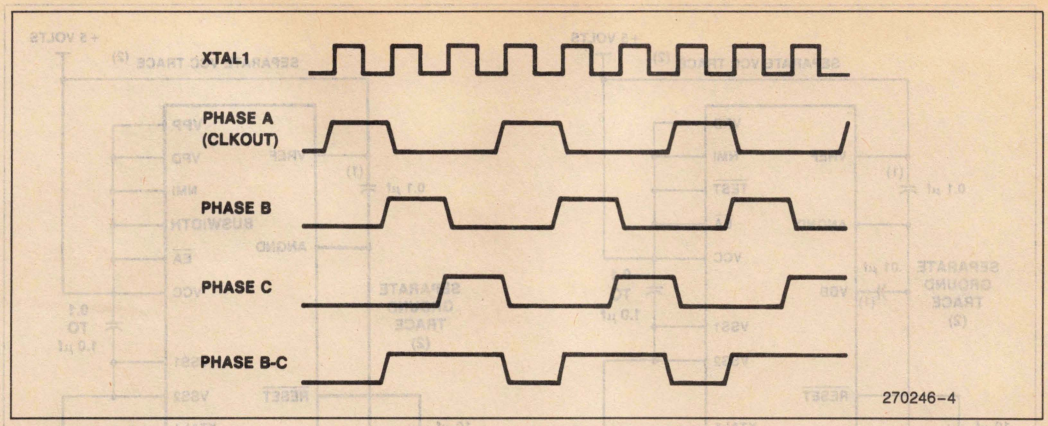


Figure 4. Internal Timings

1.4 Reset Information

In order for the 8096BH to function properly it must be reset. This is done by holding the $\overline{\text{RESET}}$ pin low for at least 2 state times after the power supply is within tolerance and the oscillator has stabilized.

On 8X9X devices the $\overline{\text{RESET}}$ pin must be held low long enough for the power supply, oscillator and back-bias generator to stabilize. Typically, the back-bias generator requires one millisecond to stabilize.

After the $\overline{\text{RESET}}$ pin is brought high, a ten state reset sequence is executed. During this time, the Chip Configuration Byte (CCB) is read from location 2018H and written to the 8096BH Chip Configuration Register (CCR). If the voltage on the $\overline{\text{EA}}$ pin selects the internal/external execution mode the CCB is read from internal ROM/EPROM. If the voltage on the $\overline{\text{EA}}$ pin selects the external execution only mode the CCB is read from external memory. See Figure 5.

On 8X9X devices, the CCB read does not occur, and ALE is high while $\overline{\text{RESET}}$ is held low.

There are several ways to provide a good reset to an 8096BH, the simplest being just to connect a capacitor from the reset pin to ground. The capacitor should be on the order of 1 to 2 microfarads for every millisecond of reset time required. This method will only work if the rise time of VCC is fast and the total reset time is less than around 50 milliseconds. It also may not work if the $\overline{\text{RESET}}$ pin is to be used to reset other parts on the board. An 8096BH with the minimum required connections is shown in Figure 6.

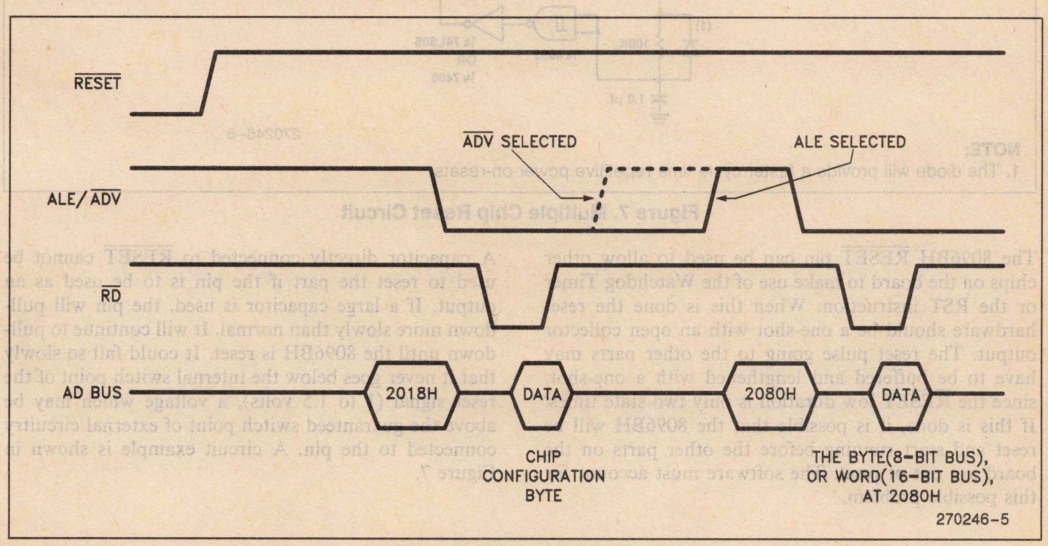
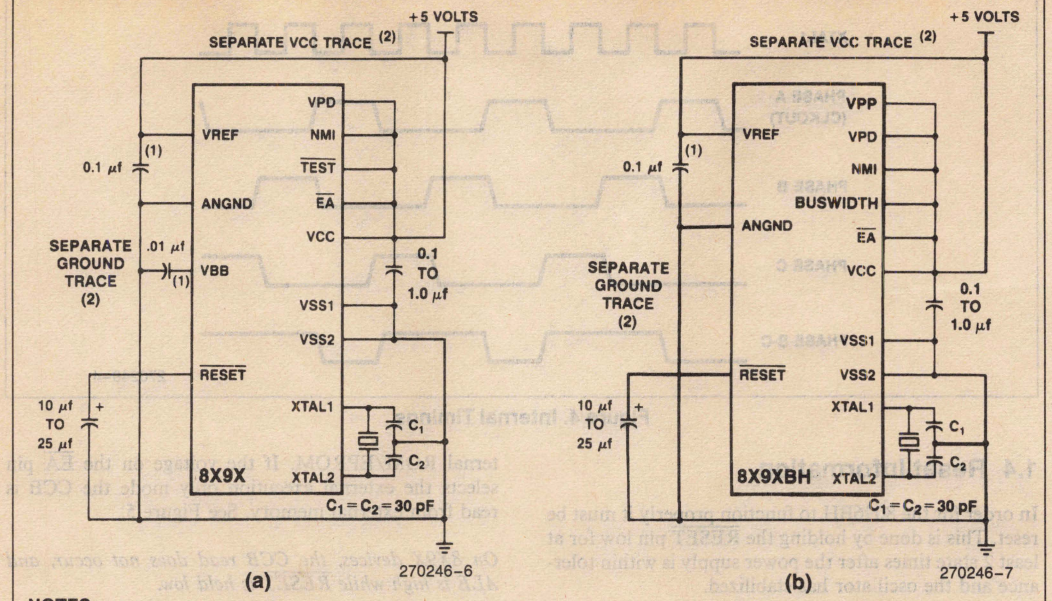
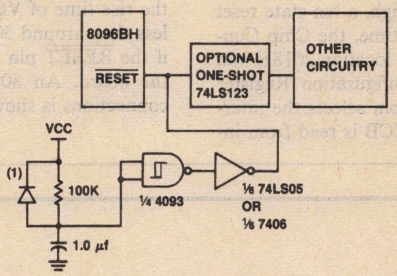


Figure 5. Reset Sequence



NOTES:
1. These capacitors are needed only if A to D is used.
2. VREF & ANGND may be connected to the same traces as the digital power supply if the A to D is not used.

Figure 6. Minimum Hardware Connections



NOTE:
1. The diode will provide a faster cycle time repetitive power-on-resets.

Figure 7. Multiple Chip Reset Circuit

The 8096BH $\overline{\text{RESET}}$ pin can be used to allow other chips on the board to make use of the Watchdog Timer or the RST instruction. When this is done the reset hardware should be a one-shot with an open collector output. The reset pulse going to the other parts may have to be buffered and lengthened with a one-shot, since the $\overline{\text{RESET}}$ low duration is only two state times. If this is done, it is possible that the 8096BH will be reset and start running before the other parts on the board are out of reset. The software must account for this possible problem.

A capacitor directly connected to $\overline{\text{RESET}}$ cannot be used to reset the part if the pin is to be used as an output. If a large capacitor is used, the pin will pull-down more slowly than normal. It will continue to pull-down until the 8096BH is reset. It could fall so slowly that it never goes below the internal switch point of the reset signal (1 to 1.5 volts), a voltage which may be above the guaranteed switch point of external circuitry connected to the pin. A circuit example is shown in Figure 7.

1.5 Sync Mode

If $\overline{\text{RESET}}$ is brought high at the same time as or just after the rising edge of XTAL1, the part will start executing the 10 state time RST instruction exactly 6½ XTAL1 cycles later. This feature can be used to synchronize several MCS-96 devices. A diagram of a typical connection is shown in Figure 8. It should be noted that parts that start in sync may not stay that way, due to propagation delays which may cause the synchronized parts to receive signals at slightly different times.

1.6 Disabling the Watchdog Timer

The Watchdog Timer will pull the $\overline{\text{RESET}}$ pin low when it overflows. See Figure 9. If the pin is being externally held above the low going threshold, the pulldown transistor will remain on indefinitely. This means that once the watchdog overflows, the part must be reset or $\overline{\text{RESET}}$ must be held high indefinitely. Just resetting the Watchdog Timer in software will not clear the flip-flop which keeps the $\overline{\text{RESET}}$ pulldown on.

The pulldown is capable of sinking on the order of 30 milliamps if it is held at 2.0 volts. This amount of current may cause some long term reliability problems due to localized chip heating. For this reason, parts that will be used in production should never have had the Watchdog Timer over-ridden for more than a second or two.

Whenever the reset pin is being pulled high while the pulldown is on, it should be through a resistor that will limit the voltage on $\overline{\text{RESET}}$ to 2.5 volts and the current through the pin to 40 milliamps.

If it is necessary to disable the Watchdog Timer for more than a brief test the software solution of never initiating the timer should be used. See Section 14 in the Architecture Chapter.

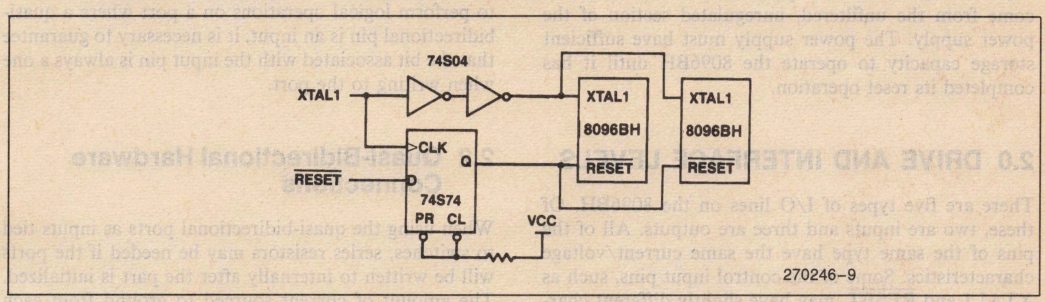


Figure 8. Reset Sync Mode

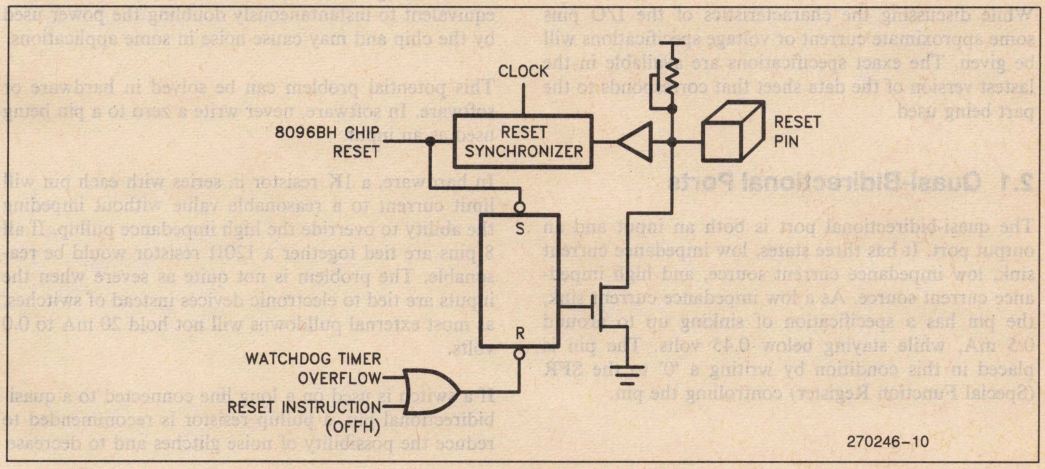


Figure 9. Reset Logic

1.7 Power Down Circuitry

Battery backup can be provided on the 8096BH with a 1 mA current drain at 5 volts. This mode will hold locations 0F0H through 0FFH valid as long as the power to the V_{PD} pin remains on. The required timings to put the part into power-down and an overview of this mode are given in Section 4.2 in the MCS-96 Architecture Chapter.

A 'key' can be written into power-down RAM while the part is running. This key can be checked on reset to determine if it is a start-up from power-down or a complete cold start. In this way the validity of the power-down RAM can be verified. The length of this key determines the probability that this procedure will work, however, there is always a statistical chance that the RAM will power up with a replica of the key.

Under most circumstances, the power-fail indicator which is used to initiate a power-down condition must come from the unfiltered, unregulated section of the power supply. The power supply must have sufficient storage capacity to operate the 8096BH until it has completed its reset operation.

2.0 DRIVE AND INTERFACE LEVELS

There are five types of I/O lines on the 8096BH. Of these, two are inputs and three are outputs. All of the pins of the same type have the same current/voltage characteristics. Some of the control input pins, such as XTAL1 and $\overline{\text{RESET}}$, may have slightly different characteristics. These pins are discussed in Section 1.

While discussing the characteristics of the I/O pins some approximate current or voltage specifications will be given. The exact specifications are available in the lastest version of the data sheet that corresponds to the part being used.

2.1 Quasi-Bidirectional Ports

The quasi-bidirectional port is both an input and an output port. It has three states, low impedance current sink, low impedance current source, and high impedance current source. As a low impedance current sink, the pin has a specification of sinking up to around 0.5 mA, while staying below 0.45 volts. The pin is placed in this condition by writing a '0' to the SFR (Special Function Register) controlling the pin.

When a '1' is written to the SFR location controlling the pin, a low impedance current source is turned on for one state time, then it is turned off and the depletion pullup holds the line at a logical '1' state. The low-impedance pullup is used to shorten the rise time of the pin, and has current source capability on the order of 100 times that of the depletion pullup. The configuration of a quasi-bidirectional port pin is shown in Figure 10.

While the depletion mode pullup is the only device on, the pin may be used as an input with a leakage of around 100 microamps from 0.45 volts to VCC. It is ideal for use with TTL or CMOS chips and may even be used directly with switches. However if the switch option is used, certain precautions should be taken. It is important to note that any time the pin is read, the value returned will be the value on the pin, not the value placed in the control register. This could cause logical operations made directly on these pins to indavertently write a 0 to pins being used as inputs. In order to perform logical operations on a port where a quasi-bidirectional pin is an input, it is necessary to guarantee that the bit associated with the input pin is always a one when writing to the port.

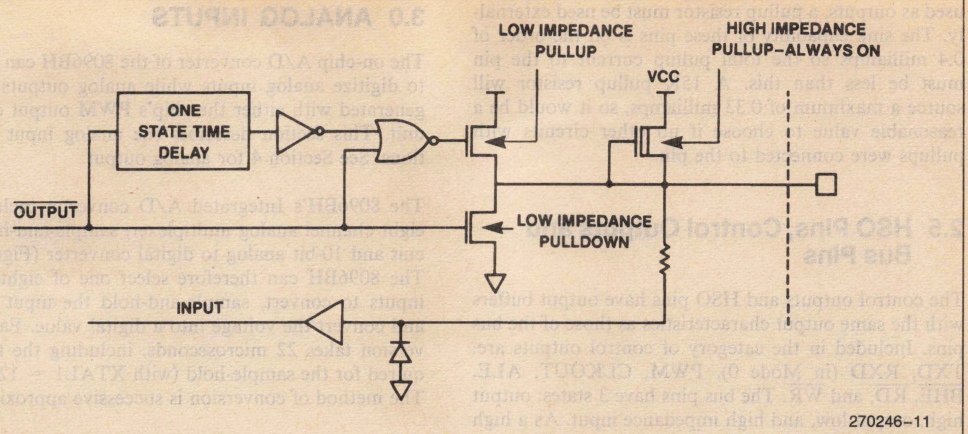
2.2 Quasi-Bidirectional Hardware Connections

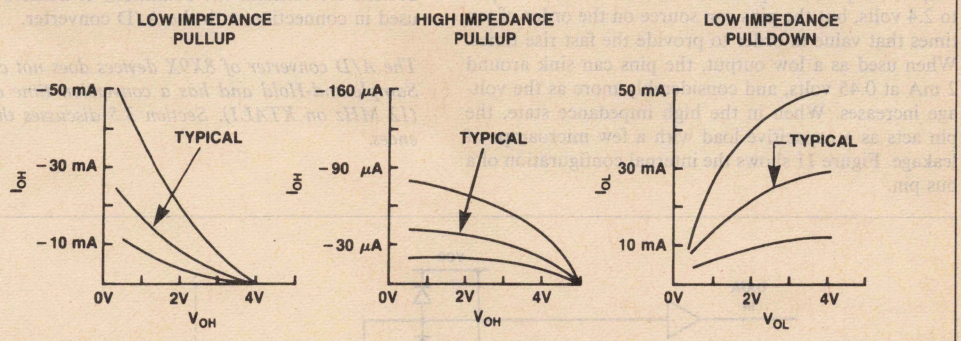
When using the quasi-bidirectional ports as inputs tied to switches, series resistors may be needed if the ports will be written to internally after the part is initialized. The amount of current sourced to ground from each pin is tyically 20 mA or more. Therefore, if all 8 pins are tied to ground, 160 mA will be sourced. This is equivalent to instantaneously doubling the power used by the chip and may cause noise in some applications.

This potential problem can be solved in hardware or software. In software, never write a zero to a pin being used as an input.

In hardware, a 1K resistor in series with each pin will limit current to a reasonable value without impeding the ability to override the high impedance pullup. If all 8 pins are tied together a 120Ω resistor would be reasonable. The problem is not quite as severe when the inputs are tied to electronic devices instead of switches, as most external pulldowns will not hold 20 mA to 0.0 volts.

If a switch is used on a long line connected to a quasi-bidirectional pin, a pullup resistor is recommended to reduce the possibility of noise glitches and to decrease





NOTE:
These graphs show typical pin capabilities, they are not guaranteed specifications

Figure 10. Quasi-Bidirectional Port

the rise time of the line. On extremely long lines that are handling slow signals, a capacitor may be helpful in addition to the resistor to reduce noise.

2.3 Input Only Ports

The high impedance input pins on the 8096BH have an input leakage of a few microamps and are predominantly capacitive loads on the order of 10 pF.

Port 0 pins are special in that they may individually be used as digital inputs, or as analog inputs. A Port 0 pin being used as a digital input acts as the high impedance input ports just described. However, Port 0 pins being used as analog inputs are required to provide current to the internal sample capacitor when a conversion begins. This means that the input characteristics of a pin will change if a conversion is being done on that pin. See Section 3.

Port 0 pins on 8X9X devices being used as analog inputs are required to provide current to an internal capacitor multiple times while a conversion is in progress. This means that the input characteristics of a Port 0 pin will change if a conversion is being done on that pin. See Section 3.

2.4 Open Drain Ports

Ports 3 and 4 on the 8096BH are open drain ports. There is no pullup when these pins are used as I/O ports. These pins have different characteristics when used as bus pins as described in the next section. A diagram of the output buffers connected to Ports 3 and 4 and the bus pins is shown in Figure 11.

When Ports 3 and 4 are to be used as inputs, or as bus pins, they must first be written with a '1'. This will put the ports in a high impedance mode. When they are

used as outputs, a pullup resistor must be used externally. The sink capability of these pins is on the order of 0.4 milliamps so the total pullup current to the pin must be less than this. A 15K pullup resistor will source a maximum of 0.33 milliamps, so it would be a reasonable value to choose if no other circuits with pullups were connected to the pin.

2.5 HSO Pins, Control Outputs and Bus Pins

The control outputs and HSO pins have output buffers with the same output characteristics as those of the bus pins. Included in the category of control outputs are: TXD, RXD (in Mode 0), PWM, CLKOUT, ALE, $\overline{BHE}$, $\overline{RD}$, and $\overline{WR}$. The bus pins have 3 states: output high, output low, and high impedance input. As a high output, the pins are specified to source around 200 μA to 2.4 volts, but the pins can source on the order of ten times that value in order to provide the fast rise times. When used as a low output, the pins can sink around 2 mA at 0.45 volts, and considerably more as the voltage increases. When in the high impedance state, the pin acts as a capacitive load with a few microamps of leakage. Figure 11 shows the internal configuration of a bus pin.

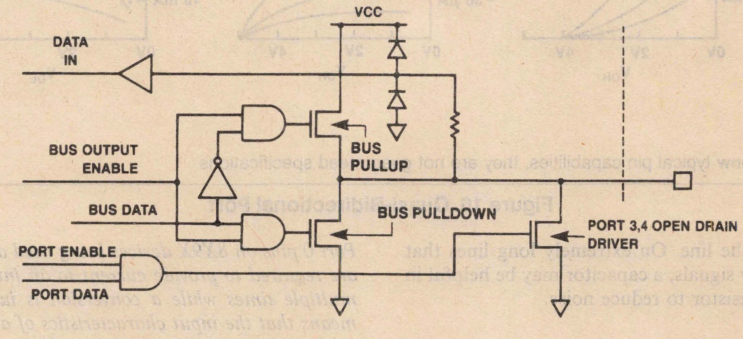
3.0 ANALOG INPUTS

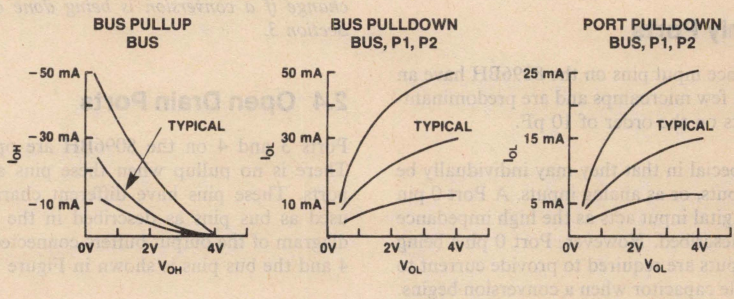
The on-chip A/D converter of the 8096BH can be used to digitize analog inputs while analog outputs can be generated with either the chip's PWM output or HSO unit. This section describes the analog input suggestions. See Section 4 for analog output.

The 8096BH's Integrated A/D converter includes an eight channel analog multiplexer, sample-and-hold circuit and 10-bit analog to digital converter (Figure 12). The 8096BH can therefore select one of eight analog inputs to convert, sample-and-hold the input voltage and convert the voltage into a digital value. Each conversion takes 22 microseconds, including the time required for the sample-hold (with XTAL1 = 12 MHz). The method of conversion is successive approximation.

Section 3.6 contains the definitions of numerous terms used in connection with the A/D converter.

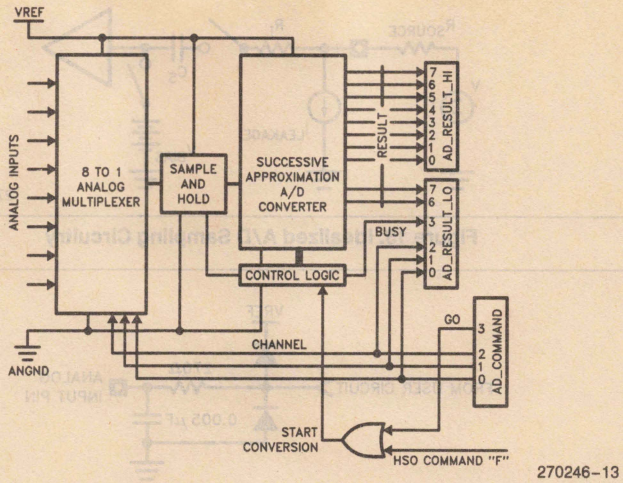
The A/D converter of 8X9X devices does not contain a Sample-and-Hold and has a conversion time of 42 μs (12 MHz on XTAL1). Section 3.5 discusses the differences.





NOTE:
These graphs show typical pin capabilities, they are not guaranteed specifications.

Figure 11. Bus and Port 3 and 4 Pins



NOTE:
1. Sample and hold not on 8X9X devices.

Figure 12. A/D Converter Block Diagram

3.1 A/D Overview

The conversion process is initiated by the execution of HSO command 0FH, or by writing a one to the GO Bit in the A/D Control Register. Either activity causes a start conversion signal to be sent to the A/D converter control logic. If an HSO command was used, the conversion process will begin when Timer 1 increments. This aids applications attempting to approach spectrally pure sampling, since successive samples spaced by equal Timer 1 delays will occur with a variance of about ±50 ns (assuming a stable clock on XTAL1). However, conversions initiated by writing a one to the ADCON register GO Bit will start within three state times after the instruction has completed execution resulting in a variance of about 0.75 μs (XTAL1 = 12 MHz).

Once the A/D unit receives a start conversion signal, there is a one state time delay before sampling (sample delay) while the successive approximation register is reset and the proper multiplexer channel is selected. After the sample delay, the multiplexer output is connected to the sample capacitor and remains connected for four state times (sample time). After this four state time "sample window" closes, the input to the sample capacitor is disconnected from the multiplexer so that changes on the input pin will not alter the stored charge while the conversion is in progress. The comparator is then auto-zeroed and the conversion begins. The sample delay and sample time uncertainties are each approximately ±50 ns, independent of clock speed.

To perform the actual analog-to-digital conversion the 8096BH implements a successive approximation algorithm. The converter hardware consists of a 256-resistor ladder, a comparator, coupling capacitors and a 10-bit successive approximation register (SAR) with logic that guides the process. The resistor ladder provides 20 mV steps (VREF = 5.12V), while capacitive coupling is used to create 5 mV steps within the 20 mV ladder voltages. Therefore, 1024 internal reference voltages are available for comparison against the analog input to generate a 10-bit conversion result.

A successive approximation conversion is performed by comparing a sequence of reference voltages, to the analog input, in a binary search for the reference voltage that most closely matches the input. The ½ full scale reference voltage is the first tested. This corresponds to a 10-bit result where the most significant bit is zero, and all other bits are ones (0111.1111.11b). If the analog input was less than the test voltage, bit 10 of the SAR is left a zero, and a new test voltage of ¼ full scale (0011.1111.11b) is tried. If this test voltage was lower than the analog input, bit 9 of the SAR is set and bit 8 is cleared for the next test (0101.1111.11b). This binary search continues until 10 tests have occurred, at which time the valid 10-bit conversion result resides in the SAR where it can be read by software.

The total number of state times required is 88 for a 10-bit conversion. Attempting to short-cycle the 10-bit conversion process by reading A/D results before the done bit is set is not recommended.

3.2 A/D Interface Suggestions

The external interface circuitry to an analog input is highly dependent upon the application, and can impact converter characteristics. In the external circuit's de-

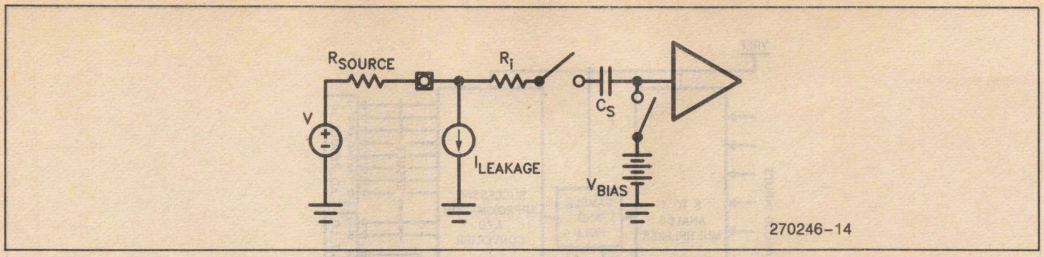


Figure 13. Idealized A/D Sampling Circuitry

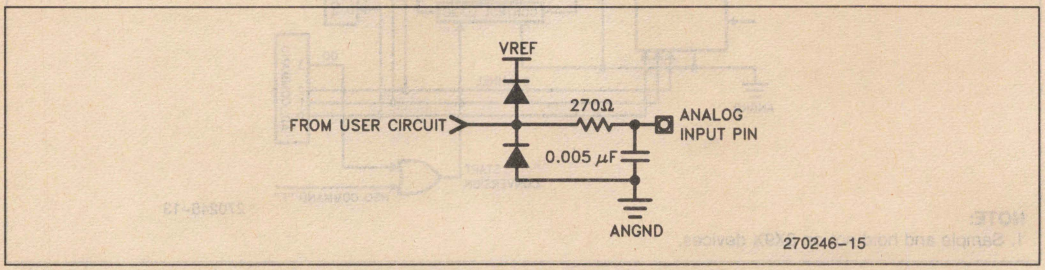


Figure 14. Suggested A/D Input Circuit

sign, important factors such as input pin leakage, sample capacitor size and multiplexer series resistance from the input pin to the sample capacitor must be considered.

For the 8096BH, these factors are idealized in Figure 13. The external input circuit must be able to charge a sample capacitor (C_S) through a series resistance (R_I) to an accurate voltage given a D.C. leakage (I_L). On the 8096BH, C_S is around 2 pF, R_I is around 5 KΩ and I_L is specified as 3 μA maximum. In determining the necessary source impedance R_S, the value of V_{BIAS} is not important.

External circuits with source impedances of 1 KΩ or less will be able to maintain an input voltage within a tolerance of about ±0.61 LSB (1.0 KΩ × 3.0 μA = 3.0 mV) given the D.C. leakage. Source impedances above 2 KΩ can result in an external error of at least one LSB due to the voltage drop caused by the 1 μA leakage. In addition, source impedances above 25 KΩ may degrade converter accuracy as a result of the internal sample capacitor not being fully charged during the 1 μs (12 MHz clock) sample window.

It is important to note that source impedance requirements relax if an external capacitor of sufficient size is attached directly to the analog input pin. Since the internal sample capacitor is around 2.0 pF, an external 0.005 μF capacitor (2048 × 2.0 pF) should provide an accurate input voltage to ±0.5 LSB. If there is leakage on the capacitor, the value of the capacitor must be increased to compensate for the leakage. For example, assuming just the 3 μA D.C. leakage caused by the 8096BH, 0.6 mV (less than 0.15 LSB) will be lost from a 0.005 μF capacitor in 1 μs. Therefore, the capacitor connected externally to the pin should be at least 0.005 μF if the source impedance is too large to provide the needed accuracy on its own. However, if the external signal changes slowly, it is recommended that the largest acceptable capacitance be used, given the input signal frequency.

Placing an external capacitor on each analog input will also reduce the sensitivity to noise, as the capacitor combines with series resistance in the external circuit to form a low-pass filter. In practice, one should include a small series resistance prior to the external capacitor on the analog input pin and choose the largest capacitor value practical, given the frequency of the signal being converted. This provides a low-pass filter on the input, while the resistor will also limit input current during over-voltage conditions.

Figure 14 shows a simple analog interface circuit based upon the discussion above. The circuit in the figure also provides limited protection against over-voltage conditions on the analog input (limits to 2.6 mA with 270Ω (0.7/270)). *However, before any circuit is used in an actual application, it should be thoroughly analyzed for applicability to the specific problem at hand.*

3.3 Analog References

Reference supply levels strongly influence the absolute accuracy of the conversion. For this reason, it is recommended that the ANGND pin be tied to a clean ground, as close to the power supply as possible. Bypass capacitors should also be used between VREF and ANGND. ANGND should be within about a tenth of a volt VSS. VREF should be well regulated and used only for the A/D converter. The VREF supply can be between 4.5V and 5.5V and needs to be able to source around 5 mA. Figure 6 shows all of these connections.

Note that if only ratiometric information is desired, VREF can be connected to VCC. In addition, VREF and ANGND must be connected even if the A/D converter is not being used.

3.4 The A/D Transfer Function

The conversion result is a 10-bit ratiometric representation of the input voltage, so the numerical value obtained from the conversion will be:

$$\text{INT}\,[1023 \times (V_{IN} - \text{ANGND})/(V_{REF} - \text{ANGND})].$$

This produces a stair-stepped transfer function when the output code is plotted versus input voltage (see Figure 15). The resulting digital codes can be taken as simple ratiometric information, or they can be used to provide information about absolute voltages or relative voltage changes on the inputs. The more demanding the application is on the A/D converter, the more important it is to fully understand the converter's operation. For simple applications, knowing the absolute error of the converter is sufficient. However, closing a servo-loop with analog inputs necessitates a detailed understanding of an A/D converter's operation and errors.

The errors inherent in an analog-to-digital conversion process are many: quantizing error; zero offset; full-scale error; differential non-linearity; and non-linearity. These are "transfer function" errors related to the A/D converter. In addition, converter temperature drift, VCC rejection, sample-hold feedthrough, multiplexer off-isolation, channel-to-channel matching and random noise should be considered. Fortunately, one "Absolute Error" specification is available which describes the sum total of all deviations between the actual conversion process and an ideal converter. However, the various sub-components of error are important in many applications. These error components are described in Section 3.5 and in the text below where ideal and actual converters are compared.

An unavoidable error simply results from the conversion of a continuous voltage to an integer digital representation. This error is called quantizing error, and is always ±0.5 LSB. Quantizing error is the only error seen in a perfect A/D converter, and is obviously present in actual converters. Figure 15 shows the transfer function for an ideal 3-bit A/D converter (i.e. the Ideal Characteristic).

Note that in Figure 15 the Ideal Characteristic possesses unique qualities: it's first code transition occurs when the input voltage is 0.5 LSB; it's full-scale code transition occurs when the input voltage equals the full-scale reference minus 1.5 LSB; and it's code widths are all exactly one LSB. These qualities result in a digitization without offset, full-scale or linearity errors. In other words, a perfect conversion.

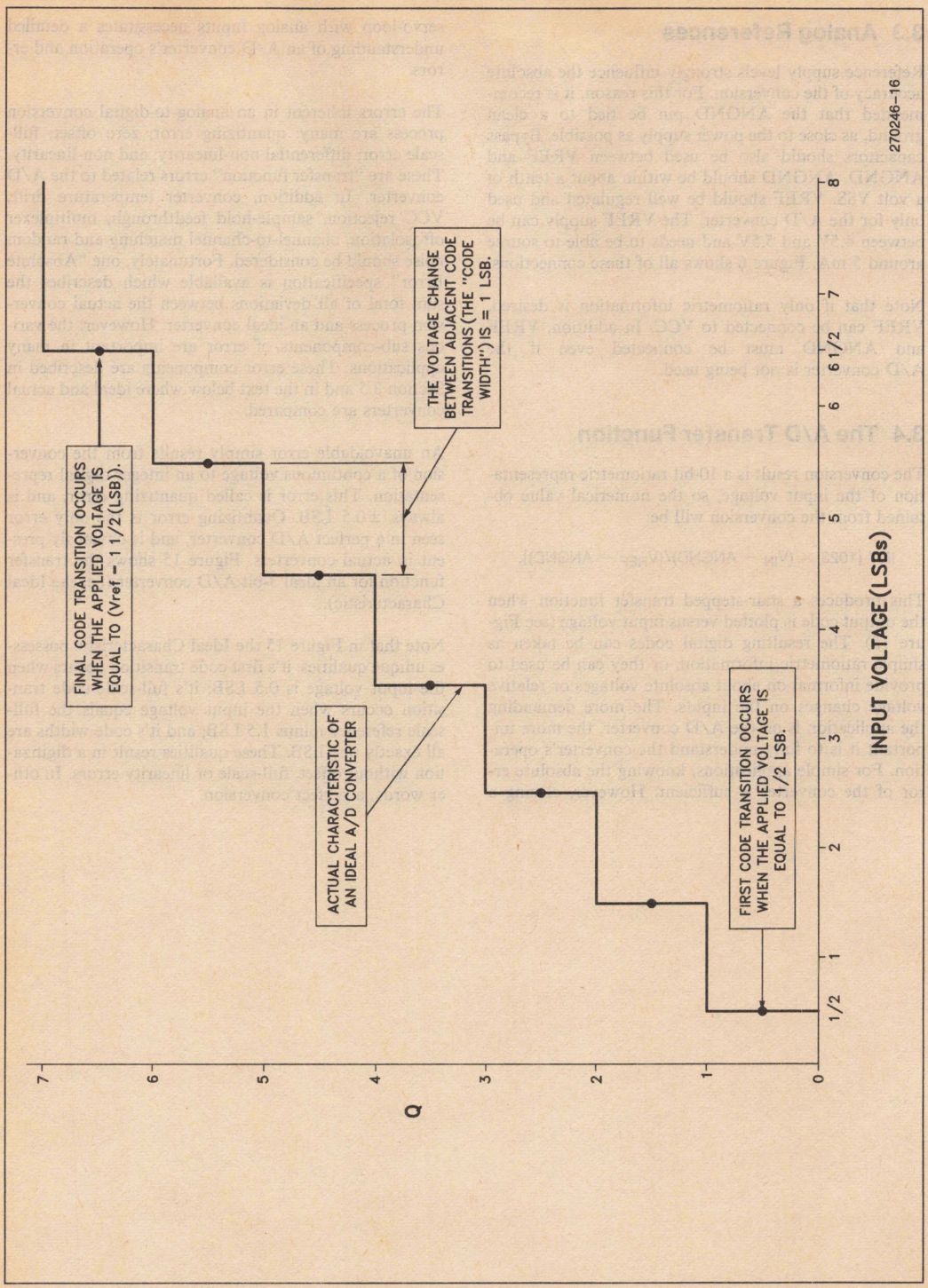


Figure 15. Ideal A/D Characteristic

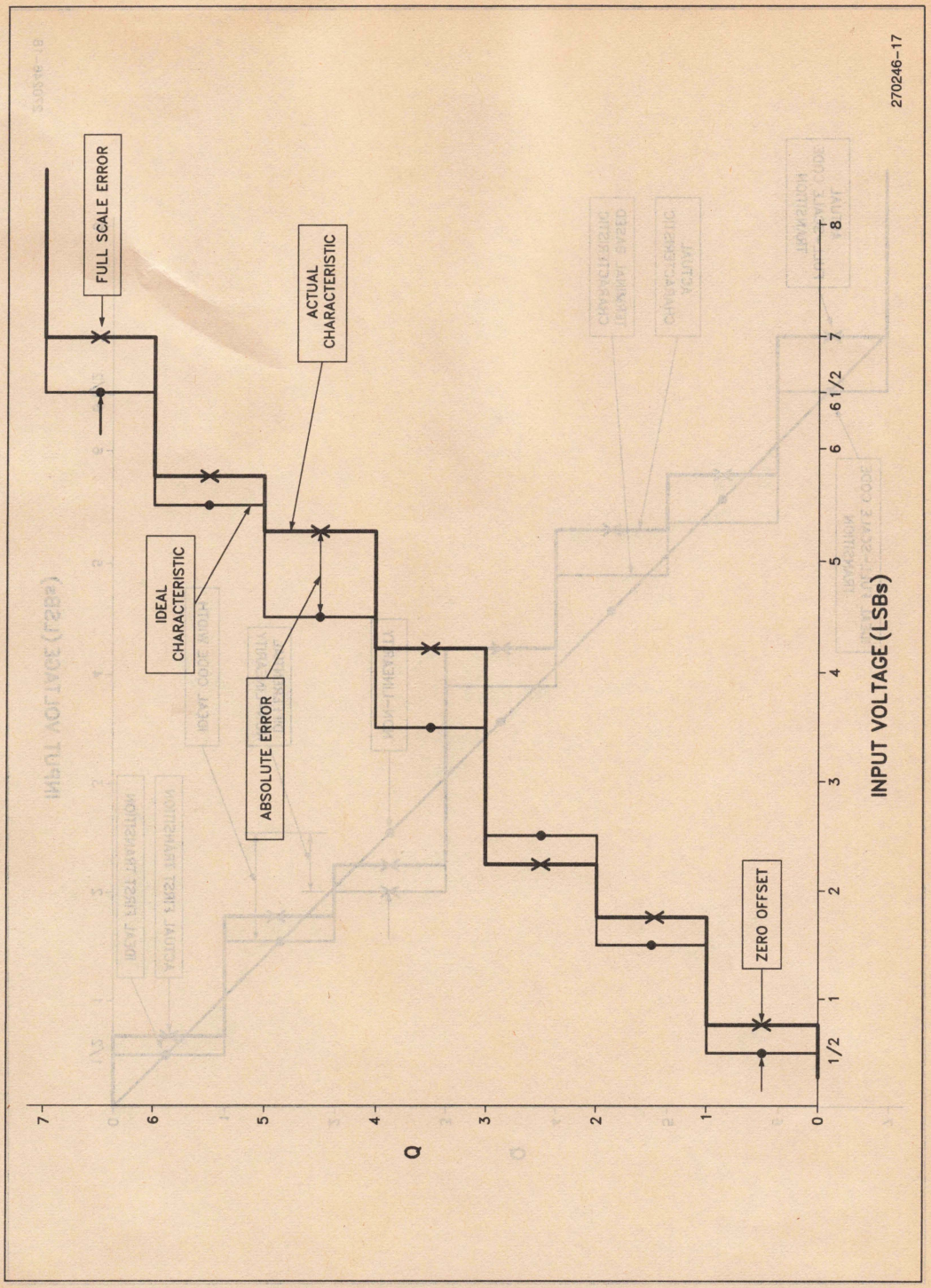


Figure 16. Actual and Ideal Characteristics

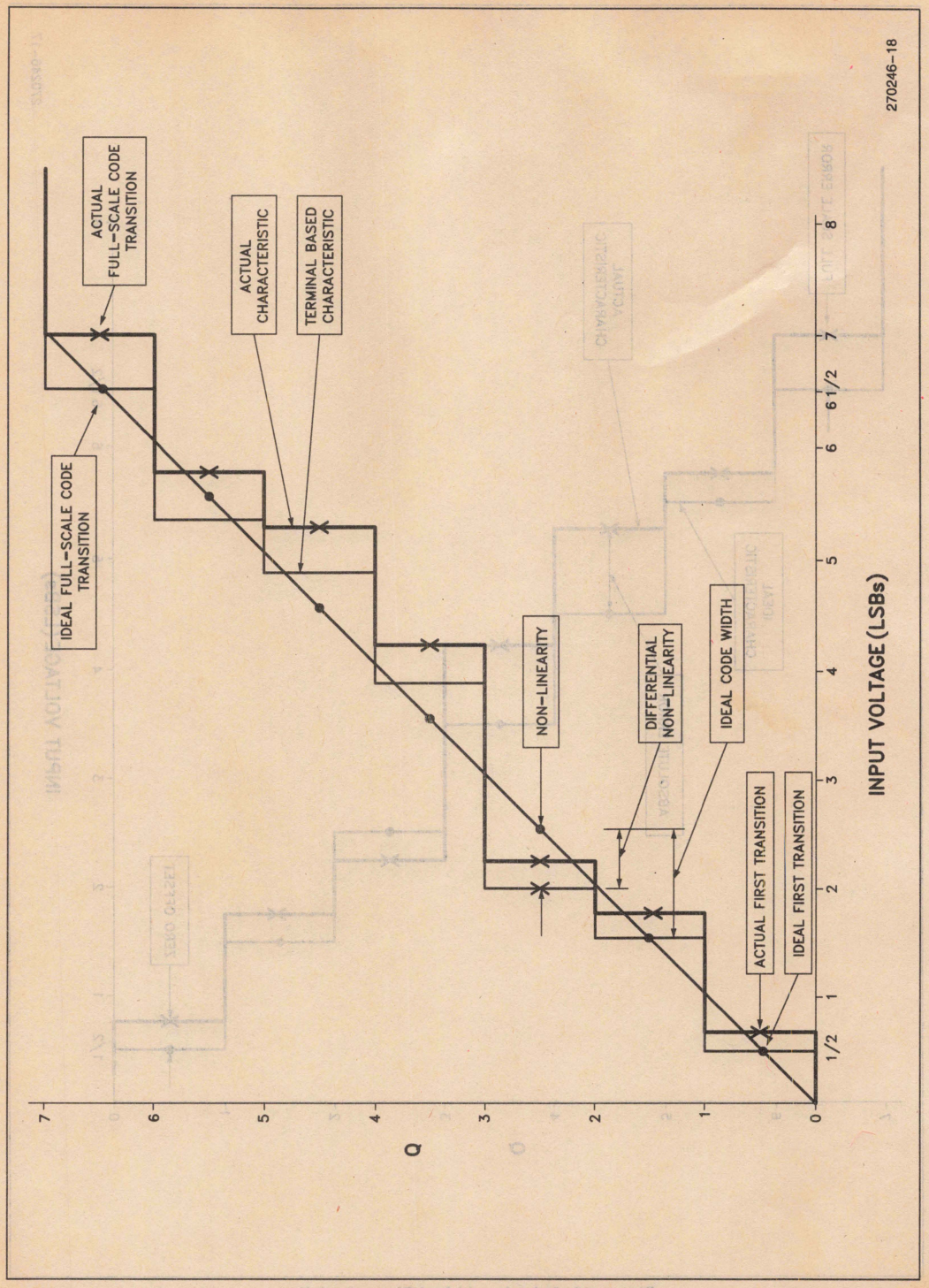


Figure 17. Terminal Based Characteristic

Figure 16 shows an Actual Characteristic of a hypothetical 3-bit converter, which is not perfect. When the Ideal Characteristic is overlaid with the imperfect characteristic, the actual converter is seen to exhibit errors in the location of the first and final code transitions and code widths. The deviation of the first code transition from ideal is called "zero offset", and the deviation of the final code transition from ideal is "full-scale error". The deviation of the code widths from ideal causes two types of errors. Differential Non-Linearity and Non-Linearity. Differential Non-Linearity is a local linearity error measurement, whereas Non-Linearity is an overall linearity error measure.

Differential Non-Linearity is the degree to which actual code widths differ from the ideal one LSB width. Differential Non-Linearity gives the user a measure of how much the input voltage may have changed in order to produce a one count change in the conversion result. Non-Linearity is the worst case deviation of code transitions from the corresponding code transitions of the Ideal Characteristic. Non-Linearity describes how much Differential Non-Linearities could add up to produce an overall maximum departure from a linear characteristic. If the Differential Non-Linearity errors are too large, it is possible for an A/D converter to miss codes or exhibit non-monotonicity. Neither behavior is desireable in a closed-loop system. A converter has no missed codes if there exists for each output code a unique input voltage range that produces that code only. A converter is monotonic if every subsequent code change represents an input voltage change in the same direction.

Differential Non-Linearity and Non-Linearity are quantified by measuring the Terminal Based Linearity Errors. A Terminal Based Characteristic results when an Actual Characteristic is shifted and rotated to eliminate zero offset and full-scale error (see Figure 17). The Terminal Based Characteristic is similar to the Actual Characteristic that would be seen if zero offset and full-scale error were externally trimmed away. In practice, this is done by using input circuits which include gain and offset trimming. In addition, VREF on the 8096BH could also be closely regulated and trimmed within the specified range to affect full-scale error.

Other factors that affect a real A/D Converter system include sensitivity to temperature, failure to completely reject all unwanted signals, multiplexer channel dissimilarities and random noise. Fortunately these effects are small.

Temperature sensitivities are described by the rate at which typical specifications change with a change in temperature.

Undesired signals come from three main sources. First, noise on VCC—VCC Rejection. Second, input signal changes on the channel being converted after the sample window has closed—Feedthrough. Third, signals applied to channels not selected by the multiplexer—Off-Isolation.

Finally, multiplexer on-channel resistances differ slightly from one channel to the next causing Channel-to-Channel Matching errors, and random noise in general results in Repeatability errors.

3.5 8X9X A/D Converter Differences

The 8X9X A/D Converter does not have an internal Sample-and-Hold, and the conversion time is 168 state times (42 μs with 12 MHz clock). These differences primarily influence the interface circuitry and the rate at which sampling can be done.

For the 8X9X, the idealized circuit in Figure 13 is still applicable. The only real difference is that the capacitor labeled C_S has a smaller value on 8X9X devices, but it is charged 10 times during a conversion. Since the actual C_S on 8X9X parts is about 0.5 pF, an effective C_S of 5.0 pF (10 × 0.5 pF) can be used as the internal capacitance that must be charged during a conversion. The value of R_I and I_L are nominally 5 kΩ and 3 μA respectively.

Given these values, external circuits with source impedances of 1 KΩ or less will be able to maintain an input voltage within a tolerance of about ±0.6 LSB (1.0 KΩ × 3.0 μA = 3.0 mV) given the D.C. leakage. Source impedances above 2 KΩ will induce an external error of at least one LSB due to the voltage drop caused by the 3 μA leakage. In addition, source impedances above 25 KΩ may degrade converter accuracy as a result of inadequate internal capacitor charging.

On 8X9X devices, the analog input is sampled 10 times while a conversion is in progress. Therefore, the input must remain stable so that conversion accuracy is not affected. If the input signal could vary significantly while a conversion is in progress, an external capacitor attached directly to the analog input pin could be used as a Sample-and-Hold. Since the internal capacitance is around 5.0 pF, an external 0.01 μF capacitor (2048 × 5.0 pF) should provide an accurate input voltage to ±0.5 LSB. If there is leakage on the capacitor, the value of the capacitor must be increased to compensate for the leakage. For example, assuming just the 3 μA D.C leakage caused by the 8X9X, 1 mV (less than 0.25 LSB) will be lost from a 0.15 μF capacitor in 42 μs. Therefore, the capacitor connected externally to the pin should be at least 0.2 μF. However, if the external signal changes slowly relative to the conversion time (168 state times), it is recommended that the largest acceptable capacitance be used given the input signal frequency.

Figure 14 shows a simple interface which could be applicable to 8X9X devices if the size of the capacitor attached to the analog input pin is increased to a value greater than 0.2 μF. The circuit in the figure also provides limited protection against over-voltage conditions on the analog inputs. However, before any circuit is used in an actual application, it should be thoroughly analyzed for applicability to the specific problem at hand.

3.6 A/D Glossary of Terms

Figures 15, 16 and 17 display many of these terms.

ABSOLUTE ERROR—The maximum difference between corresponding actual and ideal code transitions. Absolute Error accounts for all deviations of an actual converter from an ideal converter.

ACTUAL CHARACTERISTIC—The characteristic of an actual converter. The characteristic of a given converter may vary over temperature, supply voltage, and frequency conditions. An Actual Characteristic rarely has ideal first and last transition locations or ideal code widths. It may even vary over multiple conversion under the same conditions.

BREAK-BEFORE-MAKE—The property of a multiplexer which guarantees that a previously selected channel will be deselected before a new channel is selected. (e.g. the converter will not short inputs together.)

CHANNEL-TO-CHANNEL MATCHING—The difference between corresponding code transitions of actual characteristics taken from different channels under the same temperature, voltage and frequency conditions.

CHARACTERISTIC—A graph of input voltage versus the resultant output code for an A/D converter. It describes the transfer function of the A/D converter.

CODE—The digital value output by the converter.

CODE CENTER—The voltage corresponding to the midpoint between two adjacent code transitions.

CODE TRANSITION—The point at which the converter changes from an output code of Q, to a code of Q + 1. The input voltage corresponding to a code transition is defined to be that voltage which is equally likely to produce either of two adjacent codes.

CODE WIDTH—The voltage corresponding to the difference between two adjacent code transitions.

CROSSTALK—See "Off-Isolation".

D.C. INPUT LEAKAGE—Leakage current to ground from an analog input pin.

DIFFERENTIAL NON-LINEARITY—The difference between the ideal and actual code widths of the terminal based characteristic of a converter.

FEEDTHROUGH—Attenuation of a voltage applied on the selected channel of the A/D converter after the sample window closes.

FULL SCALE ERROR—The difference between the expected and actual input voltage corresponding to the full scale code transition.

IDEAL CHARACTERISTIC—A characteristic with its first code transition at VIN = 0.5 LSB, its last code transition at VIN = (VREF − 1.5 LSB) and all code widths equal to one LSB.

INPUT RESISTANCE—The effective series resistance from the analog input pin to the sample capacitor.

LSB—LEAST SIGNIFICANT BIT: The voltage value corresponding to the full scale voltage divided by 2^n, where n is the number of bits of resolution of the converter. For a 10-bit converter with a reference voltage of 5.12 volts, one LSB is 5.0 mV. Note that this is different than digital LSBs, since an uncertainty of two LSB, when referring to an A/D converter, equals 10 mV. (This has been confused with an uncertainty of two digital bits, which would mean four counts, or 20 mV.)

MONOTONIC—The property of successive approximation converters which guarantees that increasing input voltages produce adjacent codes of increasing value, and that decreasing input voltages produce adjacent codes of decreasing value.

NO MISSED CODES—For each and every output code, there exists a unique input voltage range which produces that code only.

NON-LINEARITY—The maximum deviation of code transitions of the actual characteristic from the corre-

sponding code transitions of the terminal based characteristic of the converter.

OFF-ISOLATION—Attenuation of a voltage applied on a deselected channel of the A/D converter. (Also referred to as Crosstalk.)

REPEATABILITY—The difference between corresponding code transitions from different actual characteristics taken from the same converter on the same channel at the same temperature, voltage and frequency conditions.

RESOLUTION—The number of input voltage levels that the converter can unambiguously distinguish between. Also defines the number of useful bits of information which the converter can return.

SAMPLE DELAY—The delay from receiving the start conversion signal to when the sample window opens.

SAMPLE DELAY UNCERTAINTY—The variation in the Sample Delay.

SAMPLE TIME—The time that the sample window is open.

SAMPLE TIME UNCERTAINTY—The variation in the sample time.

SAMPLE WINDOW—Begins when the sample capacitor is attached to a selected channel and ends when the sample capacitor is disconnected from the selected channel.

SUCCESSIVE APPROXIMATION—An A/D conversion method which uses a binary search to arrive at the best digital representation of an analog input.

TEMPERATURE COEFFICIENTS—Change in the stated variable per degree centigrade temperature change. Temperature coefficients are added to the typical values of a specification to see the effect of temperature drift.

TERMINAL BASED CHARACTERISTIC—An Actual Characteristic which as been rotated and translated to remove zero offset and full-scale error.

VCC REJECTION—Attenuation of noise on the VCC line to the A/D converter.

ZERO OFFSET—The difference between the expected and actual input voltage corresponding to the first code transition.

4.0 ANALOG OUTPUTS

Analog outputs can be generated by two methods, either by using the PWM output or the HSO. Either device will generate a rectangular pulse train that varies in duty cycle and (for the HSO only) period. If a smooth analog signal is desired as an output, the rectangular waveform must be filtered.

In most cases this filtering is best done after the signal is buffered to make it swing from 0 to 5 volts since both of the outputs are guaranteed only to TTL levels. A block diagram of the type of circuit needed is shown in Figure 18. By proper selection of components, accounting for temperature and power supply drift, a highly accurate 8-bit D to A converter can be made using either the HSO or the PWM output. Figure 19 shows two typical circuits. If the HSO is used the accuracy could be theoretically extended to 16-bits, however the temperature and noise related problems would be extremely hard to handle.

When driving some circuits it may be desirable to use unfiltered Pulse Width Modulation. This is particularly true for motor drive circuits. The PWM output can be used to generate these waveforms if a fixed period on the order of 64 μs is acceptable. If this is not the case then the HSO unit can be used. The HSO can generate a variable waveform with a duty cycle variable in up to 65536 steps and a period of up to 131 milliseconds. Both of these outputs produce TTL levels.

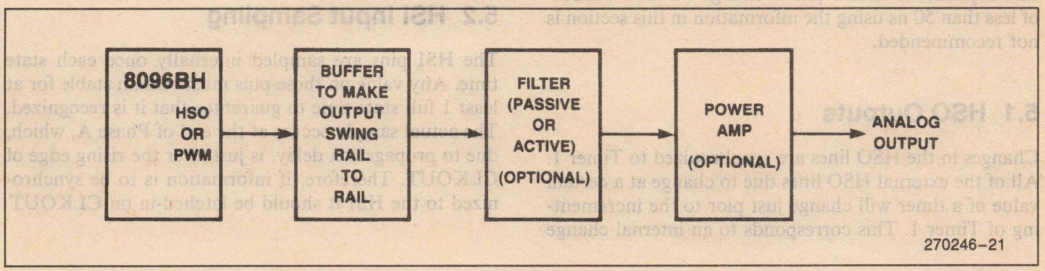


Figure 18. D/A Buffer Block Diagram

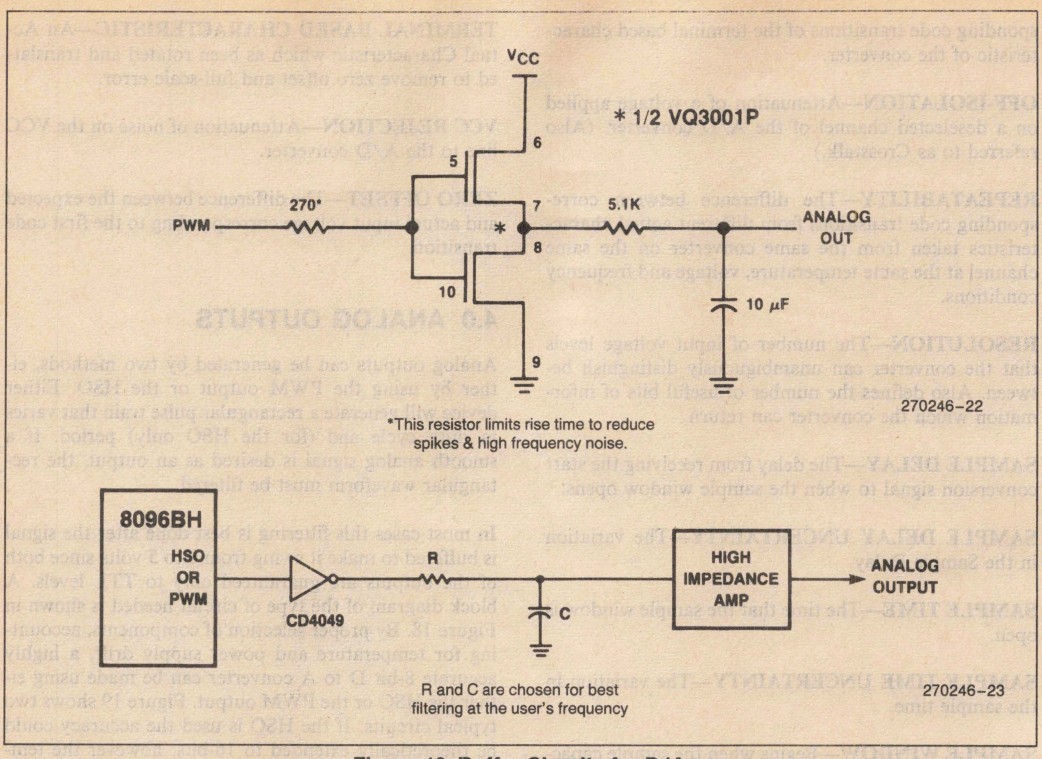


Figure 19. Buffer Circuits for D/A

5.0 I/O TIMINGS

The I/O pins on the 8096BH are sampled and changed at specific times within an instruction cycle. The changes occur relative to the internal phases shown in Figure 4. Note that the delay from XTAL1 to the internal clocks range from about 30 ns to 100 ns over process and temperature. Signals generated by internal phases are further delayed by 5 ns to 15 ns. The timings shown in this section are idealized; no propagation delay factors have been taken into account. Designing a system that depends on an I/O pin to change within a window of less than 50 ns using the information in this section is not recommended.

5.1 HSO Outputs

Changes in the HSO lines are synchronized to Timer 1. All of the external HSO lines due to change at a certain value of a timer will change just pior to the incrementing of Timer 1. This corresponds to an internal change during Phase B every eight state times. From an external perspective the HSO pin should change just prior to the rising edge of CLKOUT and be stable by its falling edge. Information from the HSO can be latched on the CLKOUT falling edge. Internal events can occur anytime during the 8 state time window.

Timer 2 is synchronized to increment no faster than Timer 1, so there will always be at least one incrementing of Timer 1 while Timer 2 is at a specific value.

5.2 HSI Input Sampling

The HSI pins are sampled internally once each state time. Any value on these pins must remain stable for at least 1 full state time to guarantee that it is recognized. The actual sample occurs at the end of Phase A, which, due to propagation delay, is just after the rising edge of CLKOUT. Therefore, if information is to be synchronized to the HSI it should be latched-in on CLKOUT

falling. The time restriction applies even if the divide by eight mode is being used. If two events occur on the same pin within the same 8 state time window, only one of the events will be recorded. If the events occur on different pins they will always be recorded, regardless of the time difference. The 8 state time window, (i.e. the amount of time during which Timer 1 remains constant), is stable to within about 20 ns. The window starts roughly around the rising edge of CLKOUT, however this timing is very approximate due to the amount of internal circuitry involved.

5.3 Standard I/O Port Pins

Port 0 is different from the other digital ports in that it is actually part of the A/D converter. The port is sampled once every state time, however, sampling is not synchronized to Timer 1. If this port is used, the input signal on the pin must be stable one state time before the reading of the SFR.

On 8X9X devices, Port 0 is sampled every eight state times (the same frequency at which the comparator is charged-up during an A/D conversion). This 8 state time counter is not synchronized with Timer 1. If this port is used, the input signal on the pin must be stable 8 state times prior to reading the SFR.

Port 1 and Port 2 have quasi-bidirectional I/O pins. When used as inputs the data on these pins must be stable one state time prior to reading the SFR. This timing is also valid for the input-only pins of Port 2 and is similar to the HSI in that the sample occurs just after the rising edge of CLKOUT. When used as outputs, the quasi-bidirectional pins will change state shortly after CLKOUT falls. If the change was from '0' to a '1' the low impedance pullup will remain on for one state time after the change.

Ports 3 and 4 are addressed as off-chip memory-mapped I/O. The port pins will change state shortly after the rising edge of CLKOUT. When these pins are used as Ports 3 and 4 they are open drains, their structure is different when they are used as part of the bus. See Section 12.4 of the MCS-96 Architecture chapter.

6.0 SERIAL PORT TIMINGS

The serial port on the 8096BH was designed to be compatible with the 8051 serial port. Since the 8051 uses a divide by 2 clock and the 8096BH uses a divide by 3, the serial port on the 8096BH had to be provided with its own clock circuit to maximize its compatibility with the 8051 at high baud rates. This means that the serial port itself does not know about state times. There is circuitry which is synchronized to the serial port and to the rest of the 8096BH so that information can be passed back and forth.

The baud rate generator is clocked by either XTAL1 or T2CLK. Because T2CLK needs to be synchronized to the XTAL1 signal its speed must be limited to $1/16$ that of XTAL1. The serial port will not function during the time between the consecutive writes to the baud rate register. Section 11.4 of the MCS-96 Architecture chapter discusses programming the baud rate generator.

6.1 Mode 0

Mode 0 is the shift register mode. The TXD pin sends out a clock train, while the RXD pin transmits or receives the data. Figure 20 shows the waveforms and timing. Note that the port starts functioning when a '1' is written to the REN (Receiver Enable) bit in the serial port control register. If REN is already high, clearing the RI flag will start a reception.

In this mode the serial port can be used to expand the I/O capability of the 8096BH by simply adding shift registers. A schematic of a typical circuit is shown in Figure 21. This circuit inverts the data coming in, so it must be reinverted in software. The enable and latch connections to the shift registers can be driven by decoders, rather than directly from the low speed I/O ports, if the software and hardware are properly designed.

6.2 Mode 1 Timings

Mode 1 operation of the serial port makes use of 10-bit data packages, a start bit, 8 data bits and a stop bit. The transmit and receive functions are controlled by separate shift clocks. The transmit shift clock starts when the baud rate generator is initialized, the receive shift clock is reset when a '1 to 0' transition (start bit) is received. The transmit clock may therefore not be in sync with the receive clock, although they will both be at the same frequency.

The TI (Transmit Interrupt) and RI (Receive Interrupt) flags are set to indicate when operations are complete. TI is set when the last data bit of the message has been sent, not when the stop bit is sent. If an attempt to send another byte is made before the stop bit is sent the port will hold off transmission until the stop bit is complete. RI is set when 8 data bits are received, not when the stop bit is received. Note that when the serial port status register is read both TI and RI are cleared.

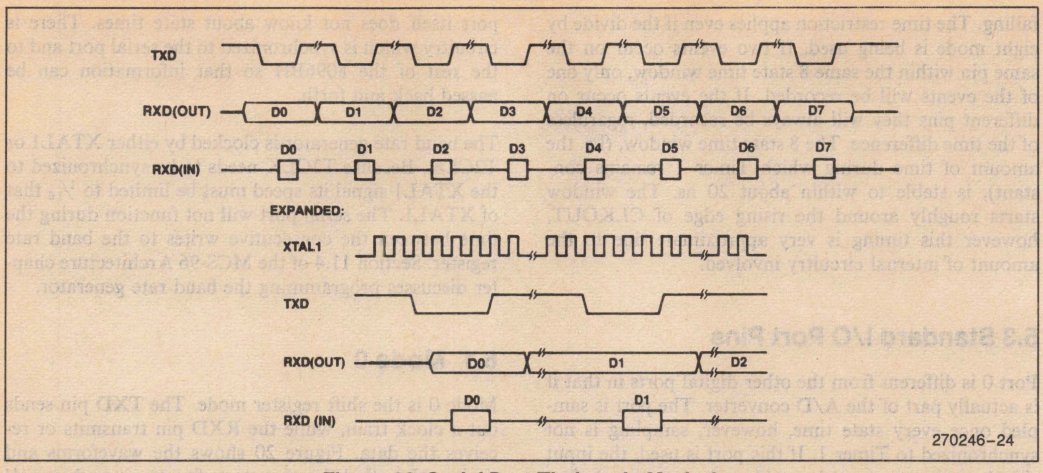


Figure 20. Serial Port Timings in Mode 0

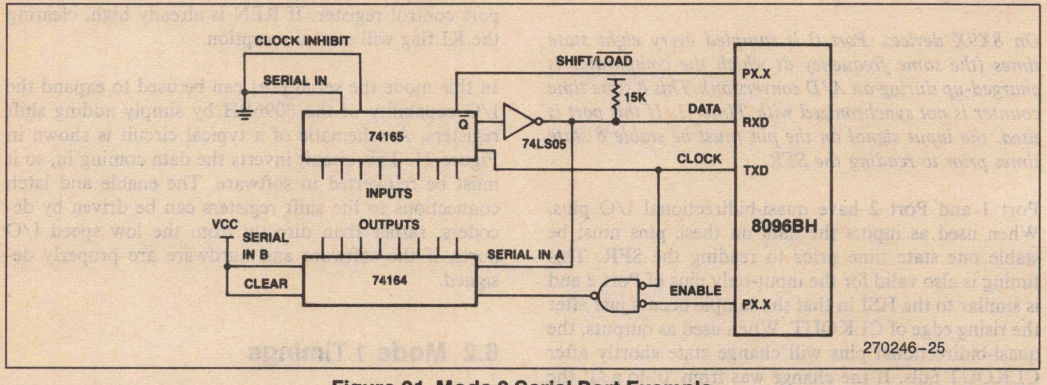


Figure 21. Mode 0 Serial Port Example

Caution should be used when using the serial port to connect more than two devices in half-duplex, (i.e. one wire for transmit *and* receive). If the receiving processor does not wait for one bit time after RI is set before starting to transmit, the stop bit on the link could be squashed. This could cause a problem for other devices listening on the link.

6.3 Mode 2 and 3 Timings

Modes 2 and 3 operate in a manner similar to that of Mode 1. The only difference is that the data is now made up of 9 bits, so 11-bit packages are transmitted and received. This means that TI and RI will be set on the 9th data bit rather than the 8th. The 9th bit can be used for parity or multiple processor communications (see Section 11 of the MCS-96 Architecture chapter).

7.0 BUS TIMING AND MEMORY INTERFACE

7.1 Bus Functionality

The 8096BH has a multiplexed (address/data) bus which can be dynamically configured to have an 8-bit or 16-bit data width. There are control lines to demultiplex the bus ($\overline{ALE}$ or $\overline{ADV}$), indicate reads ($\overline{RD}$), indicate writes ($\overline{WRL}$ and $\overline{WRH}$, or $\overline{WR}$ with $\overline{BHE}$ and AD0), and a signal to indicate accesses that are for an instruction fetch (INST). Section 3.5 of the MCS-96 Architecture chapter contains an overview of the bus operation.

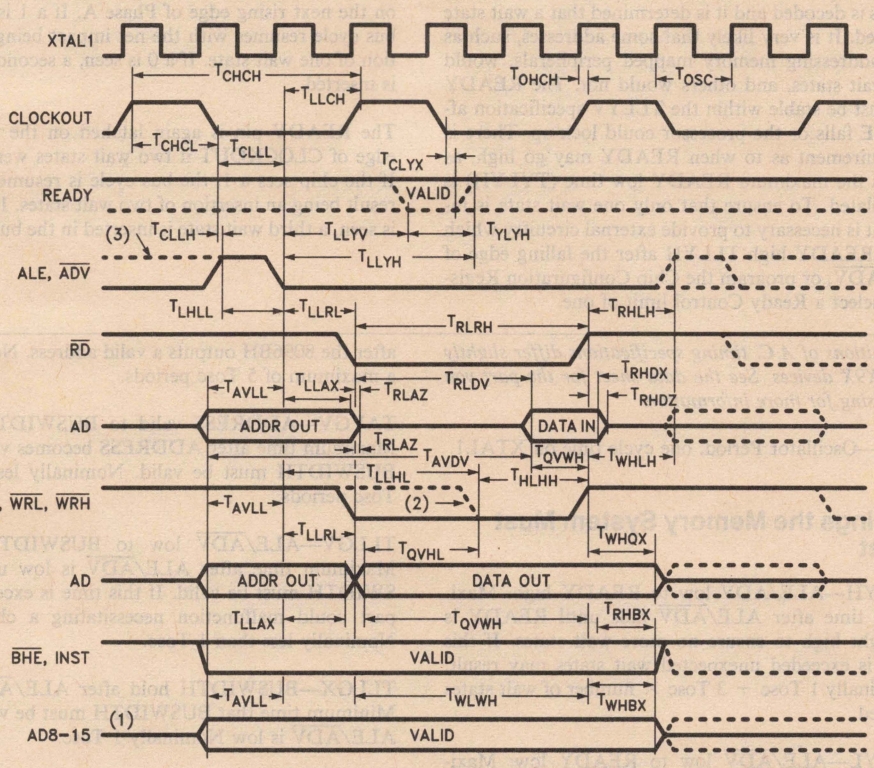
On 8X9X devices only the 16-bit multiplexed bus is available. In addition, on 8X9X devices the $\overline{WRL}$ and $\overline{WRH}$ signals are not available and the functionality of the $\overline{BHE}$ and INST lines differs from the 8X9XBH devices. See the data sheet of the device that you use.

7.2 Timing Specifications

Figure 22 shows the timing of the bus signals and data lines. Please refer to the latest data sheet for the exact device you are using to ensure that your system is designed to the proper specifications. The major timing specifications are described in Figure 23.

7.3 READY Line Usage

When the processor has to address a memory location that cannot respond within the standard specifications, it is necessary to use the READY line to generate wait states. When the READY line is held low, the processor waits in a loop for the line to come high or until the



NOTES:
1. When ALE function is selected, the signal is always high for TLHLL. When $\overline{ADV}$ function is selected, the signal is high for at least TLHLL.
2. The dotted line applies for all 8-bit bus writes and 16-bit bus writes with the write strobe mode selected.
3. 8-bit bus only.

Figure 22. Bus Signal Timings

number of inserted wait states is equal to the limit set in the Chip Configuration Register (see Section 2 of the MCS-96 Architecture chapter). There is a maximum time that the READY line can be held low without risking a processor malfunction due to dynamic nodes that have not been refreshed during the wait states. This time is shown as TYLYH in the data sheet.

In most cases the READY line is brought low after the address is decoded and it is determined that a wait state is needed. It is very likely that some addresses, such as those addressing memory mapped peripherals, would need wait states, and others would not. The READY line must be stable within the TLLYV specification after ALE falls or the processor could lock-up. There is no requirement as to when READY may go high, as long as the maximum READY low time (TYLYH) is not violated. To ensure that only one wait state is inserted it is necessary to provide external circuitry which brings READY high TLLYH after the falling edge of ALE/$\overline{\text{ADV}}$, or program the Chip Configuration Register to select a Ready Control limit of one.

Internally, the chip latches READY on the first falling edge of Phase A after ALE/$\overline{\text{ADV}}$ falls. Phase A is buffered and brought out externally as CLOCKOUT, so CLOCKOUT is a delayed Phase A. If a 1 is seen, the bus cycle proceeds uninterrupted with no wait state insertions. If a 0 is seen, one wait state (3 Tosc) is inserted.

If a wait state is inserted, READY is internally latched on the next rising edge of Phase A. If a 1 is found the bus cycle resumes with the net impact being the insertion of one wait state. If a 0 is seen, a second wait state is inserted.

The READY pin is again latched on the next rising edge of CLOCKOUT if two wait states were inserted. If the chip sees a 1, the bus cycle is resumed with the result being an insertion of two wait states. If another 0 is seen, a third wait state is inserted in the bus cycle and

Definitions of A.C. timing specifications differ slightly on 8X9X devices. See the data sheet for the part you are using for more information.

Tosc—Oscillator Period, one cycle time on XTAL1.

Timings the Memory System Must Meet

TLLYH—ALE/$\overline{\text{ADV}}$ low to READY high: Maximum time after ALE/$\overline{\text{ADV}}$ falls until READY is brought high to ensure no more wait states. If this time is exceeded unexpected wait states may result. Nominally 1 Tosc + 3 Tosc × number of wait states desired.

TLLYL—ALE/$\overline{\text{ADV}}$ low to READY low: Maximum time after ALE/$\overline{\text{ADV}}$ falls until READY must be valid. If this time is exceeded the part could malfunction necessitating a chip reset. Nominally 2 Tosc periods.

TCLYX—READY hold after CLOCKOUT low: Minimum time that the value on the READY pin must be valid after CLOCKOUT falls. The minimum hold time is always zero nanoseconds.

TYLYH—READY low to READY high: Maximum time the part can be in the not-ready state. If it is exceeded, the 8096BH dynamic nodes which hold the current instruction may 'forget' how to finish the instruction.

TAVDV—ADDRESS valid to DATA valid: Maximum time that the memory has to output valid data after the 8096BH outputs a valid address. Nominally, a maximum of 5 Tosc periods.

TAVGV—ADDRESS valid to BUSWIDTH valid: Maximum time after ADDRESS becomes valid until BUSWIDTH must be valid. Nominally less than 2 Tosc periods.

TLLGV—ALE/$\overline{\text{ADV}}$ low to BUSWIDTH valid: Maximum time after ALE/$\overline{\text{ADV}}$ is low until BUSWIDTH must be valid. If this time is exceeded the part could malfunction necessitating a chip reset. Nominally less than 1 Tosc.

TLLGX—BUSWIDTH hold after ALE/$\overline{\text{ADV}}$ low: Minimum time that BUSWIDTH must be valid after ALE/$\overline{\text{ADV}}$ is low Nominally 1 Tosc.

TRLDV—$\overline{\text{READ}}$ low to DATA valid: Maximum time that the memory has to output data after $\overline{\text{READ}}$ goes low. Nominally, a maximum of 3 Tosc periods.

TRHDZ—$\overline{\text{READ}}$ high to DATA float: Time after $\overline{\text{READ}}$ is high until the memory must float the bus. The memory signal can be removed as soon as $\overline{\text{READ}}$ is not low, and must be removed within the specified maximum time from when $\overline{\text{READ}}$ is high. Nominally a maximum of 1 Tosc period.

TRHDX—DATA hold after $\overline{\text{READ}}$ goes high: Minimum time that memory must hold input DATA valid after $\overline{\text{RD}}$ is high. The hold time minimum is always zero nanoseconds.

Figure 23. Timing Specification Explanations

Timings the 8096 Will Provide

TOHCH—XTAL1 high to CLOCKOUT high: Delay from the rising edge of XTAL1 to the resultant rising edge on CLOCKOUT. Needed in systems where the signal driving XTAL1 is also used as a clock for external devices. Typically 50 to 100 nanoseconds.

TCHCH—CLKOUT high to CLKOUT high: The period of CLKOUT and the duration of one state time. Always 3 Tosc average, but individual periods could vary by a few nanoseconds.

TCHCL—CLKOUT high to CLKOUT low: Nominally 1 Tosc period.

TCLLH—CLKOUT low to ALE high: A help in deriving other timings. Typically plus or minus 5 ns to 10 ns.

TCLVL—CLOCKOUT low to ALE/$\overline{\text{ADV}}$ low: A help in deriving other timings. Nominally 1 Tosc.

TLLCH—ALE/$\overline{\text{ADV}}$ low to CLKOUT high: Used to derive other timings, nominally 1 Tosc period.

TLHLL—ALE/$\overline{\text{ADV}}$ high to ALE/$\overline{\text{ADV}}$ low: ALE/$\overline{\text{ADV}}$ high time. Useful in determining ALE/$\overline{\text{ADV}}$ rising edge to ADDRESS valid time. Nominally 1 Tosc period for ALE and 1 Tosc for $\overline{\text{ADV}}$ with back-to-back bus cycles.

TAVLL—ADDRESS valid to ALE/$\overline{\text{ADV}}$ low: Length of time ADDRESS is valid before ALE/$\overline{\text{ADV}}$ falls. Important timing for address latch circuitry. Nominally 1 Tosc period.

TLLAX—ALE/$\overline{\text{ADV}}$ low to ADDRESS invalid: Length of time ADDRESS is valid after ALE/$\overline{\text{ADV}}$ falls. Important timing for address latch circuitry. Nominally 1 Tosc period.

TLLRL—ALE/$\overline{\text{ADV}}$ low to $\overline{\text{READ}}$ or $\overline{\text{WRITE}}$ low: Length of time after ALE/$\overline{\text{ADV}}$ falls before $\overline{\text{RD}}$ or $\overline{\text{WR}}$ fall. Could be needed to ensure that proper memory decoding takes place before it is output enabled. Nominally 1 Tosc period.

TLLHL—ALE/$\overline{\text{ADV}}$ low to $\overline{\text{WRL}}$, $\overline{\text{WRH}}$ low: Minimum time after ALE/$\overline{\text{ADV}}$ is low that the write strobe signals will go low. Could be needed to ensure that proper memory decoding takes place before it is output enabled. Nominally 2 Tosc periods.

TRLRH—$\overline{\text{READ}}$ low to $\overline{\text{READ}}$ high: $\overline{\text{RD}}$ pulse width, nominally 1 Tosc period.

TRHLH—$\overline{\text{READ}}$ high to ALE/$\overline{\text{ADV}}$ high: Time between $\overline{\text{RD}}$ going inactive and next ALE/$\overline{\text{ADV}}$, also used to calculate time between $\overline{\text{RD}}$ inactive and next ADDRESS valid. Nominally 1 Tosc period.

TRHBX—$\overline{\text{READ}}$ high to INST, $\overline{\text{BHE}}$, AD8–15 Inactive: Minimum time that the INST and BHE lines will be valid after $\overline{\text{RD}}$ goes high. Also the minimum time that the upper eight address lines (8-bit bus mode) will remain valid after $\overline{\text{RD}}$ goes high. Nominally 1 Tosc.

TWHBX—$\overline{\text{WRITE}}$ high to INST, $\overline{\text{BHE}}$, AD8–15 Inactive: Minimum time that the INST and $\overline{\text{BHE}}$ lines will be valid after $\overline{\text{WR}}$ goes high. Also the minimum time that the upper eight address lines (8-bit bus mode) will remain valid after $\overline{\text{WR}}$ goes high. Nominally 1 Tosc.

TWLWH—$\overline{\text{WRITE}}$ low to $\overline{\text{WRITE}}$ high: Write pulse width, nominally 3 Tosc periods.

THLHH—$\overline{\text{WRL}}$, $\overline{\text{WRH}}$ low to $\overline{\text{WRL}}$, $\overline{\text{WRH}}$ high: Write strobe signal pulse width. Nominally 2 Tosc periods.

TQVHL—OUTPUT valid to $\overline{\text{WRL}}$, $\overline{\text{WRH}}$ low: Minimum time that OUTPUT data is valid prior to write strobes becoming active. Needed for interfacing to memories that read data on the falling edge of write. Nominally 1 Tosc.

TQVWH—OUTPUT valid to $\overline{\text{WRITE}}$ high: Time that the OUTPUT data is valid before $\overline{\text{WR}}$ is high. Nominally 3 Tosc periods.

TWHQX—$\overline{\text{WRITE}}$ high to OUTPUT not valid: Time that the OUTPUT data is valid after $\overline{\text{WR}}$ is high. Nominally 1 Tosc period.

TWHLH—$\overline{\text{WRITE}}$ high to ALE/$\overline{\text{ADV}}$ high: Time between write high and next ALE/$\overline{\text{ADV}}$, also used to calculate the time between $\overline{\text{WR}}$ high and next ADDRESS valid. Nominally 2 Tosc periods.

Figure 23. Timing Specification Explanations (Continued)

the READY pin is again latched on the following rising edge of CLOCKOUT. If internal Ready Control is not used, the READY line must at this point be a 1 to ensure proper operation.

On 8X9X devices there is no internal Ready Control, therefore, external circuitry must completely control the insertion of wait states into 8X9X bus cycles.

7.4 INST Line Usage

The INST (Instruction) line is high during bus cycles that are for an instruction fetch and low for any other bus cycle. The INST signal (not present on 48-pin versions) can be used with a logic analyzer to debug a system. In this way it is possible to determine if a fetch was for instructions or data, making the task of tracing the program much easier.

On 8X9X devices the INST line is high during the output of an address that is for an instruction fetch. It is low during the same time for any other memory access. At any other time it is not valid.

7.5 BUSWIDTH Pin Usage

The BUSWIDTH pin is a control input which determines the width of the bus access in progress. BUSWIDTH is sampled after the rising edge of the first CLOCKOUT after ALE/$\overline{\text{ADV}}$ goes low. If a one is seen, the bus access progresses as a 16-bit cycle. If a zero is seen, the bus access progresses as an 8-bit cycle. The BUSWIDTH setup and hold timing requirements appear in the data sheet.

The BUSWIDTH pin can be overridden by causing the BUS WIDTH SELECT bit in the Chip Configuration Register (CCR) to be zero. This will permanently select an 8-bit bus width. However, if the BUS WIDTH SELECT bit in the CCR is a one, the BUSWIDTH pin determines the bus width. See Section 3.5 of the MCS-96 Architecture chapter. Since the BUSWIDTH pin is not available on 48-pin parts, the BUS WIDTH SELECT bit in the CCR determines bus width.

On 8X9X devices, the 8-bit bus is not available, the CCR does not exist and the BUSWIDTH pin is named the TEST pin. The TEST pin is used for testing purposes and should be tied to VCC in application circuits.

7.6 Address Decoding

The multiplexed bus of the 8096BH must be demultiplexed before it can be used. This can be done with two 74LS373 transparent latches for an 8096BH in 16-bit bus mode, or one 74LS373 for an 8096BH in 8-bit bus mode. As explained in Section 3.5 of the MCS-96 Architecture chapter, the latched address signals will be referred to as MA0 through MA15 (Memory Address), and the data lines will be called MD0 through MD15 (Memory Data).

Since the 8096BH can make accesses to memory for either bytes or words, it is necessary to have a way of determining the type of access desired when the bus is 16-bits wide. For write cycles, the signals Write Low ($\overline{\text{WRL}}$) and Write High ($\overline{\text{WRH}}$) are provided. $\overline{\text{WRL}}$ will go low during all word writes and during all byte writes to an even location. Similarly, $\overline{\text{WRH}}$ will go low during all word writes and during all byte writes to an odd location. During read cycles, an 8096BH in 16-bit bus mode will always do a word read of an even location. If only one byte of the word is needed, the chip discards the byte it does not need.

Since 8096BH memory accesses over an 8-bit wide bus are always bytes, only one write strobe is needed for write cycles. For this purpose the $\overline{\text{WRL}}$ signal was made to go low for all write cycles during 8-bit bus accesses. When a word operation is requested, the bus controller performs two byte-wide bus cycles.

In many cases it may be desirable to have a write signal with a longer pulse width than $\overline{\text{WRL}}$/$\overline{\text{WRH}}$. The Write ($\overline{\text{WR}}$) line of the 8096BH is an alternate control signal that shares a pin with $\overline{\text{WRL}}$ and is only available in 16-bit bus mode. $\overline{\text{WR}}$ is nominally one Tosc longer than the $\overline{\text{WRL}}$/$\overline{\text{WRH}}$ signals, but goes low for any write cycle. Therefore it is necessary to decode for the type of write (byte or word) desired.

The Byte High Enable ($\overline{\text{BHE}}$) signal and MA0 can be used for this purpose. $\overline{\text{BHE}}$ is an alternate control sig-

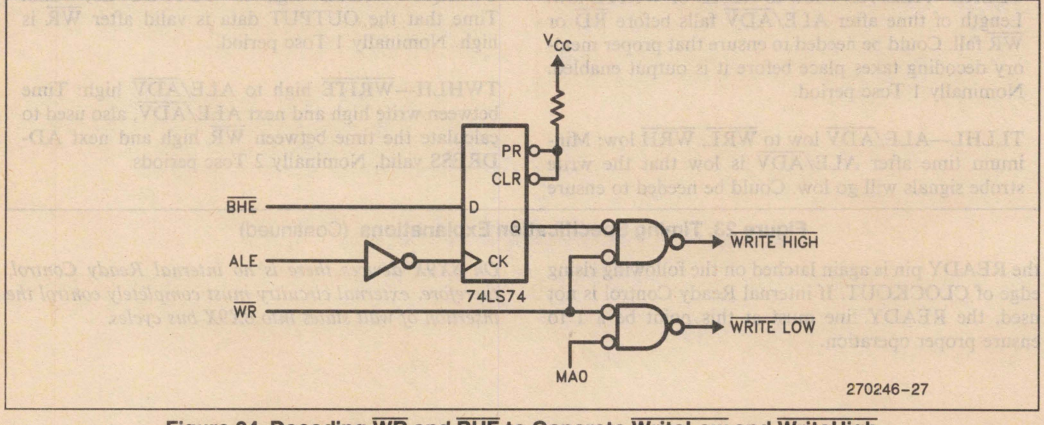


Figure 24. Decoding $\overline{\text{WR}}$ and $\overline{\text{BHE}}$ to Generate $\overline{\text{WriteLow}}$ and $\overline{\text{WriteHigh}}$

nal that shares a pin with $\overline{WRH}$. When $\overline{BHE}$ is low, the high byte of the 16-bit bus is enabled. When MA0 is low, the lower byte is enabled. When MA0 is low and $\overline{BHE}$ is low, both bytes are enabled. Figure 24 shows how to use $\overline{WR}$, $\overline{BHE}$ and $\overline{MA0}$ to decode bus accesses. It's important to note that this decoding inserts a delay in the write signal which must be considered in a system timing analysis.

On 8X9X devices, only the $\overline{RD}$, $\overline{WR}$ and $\overline{BHE}$ signals are available for bus control. This means that discriminating between byte and word bus accesses must be done by decoding $\overline{WR}$, $\overline{BHE}$ and MA0 as described above.

Further, the $\overline{WR}$ signal on 8X9X devices is nominally the same width as the $\overline{WRL}$ and $\overline{WRH}$ signals. 8X9XBH devices (2 Tosc), and the $\overline{BHE}$ signal must be latched since it is valid only while the address is valid. See Figure 24 and the data sheet of the device that you use.

External memory systems for the 8096BH can be set up in many ways. Figures 25 through 28 show block diagrams of memory systems using an 8-bit bus with a single EPROM, using an 8-bit bus with RAM and EPROM, using a 16-bit bus with two external EPROMs and using a 16-bit bus in a RAM and ROM system.

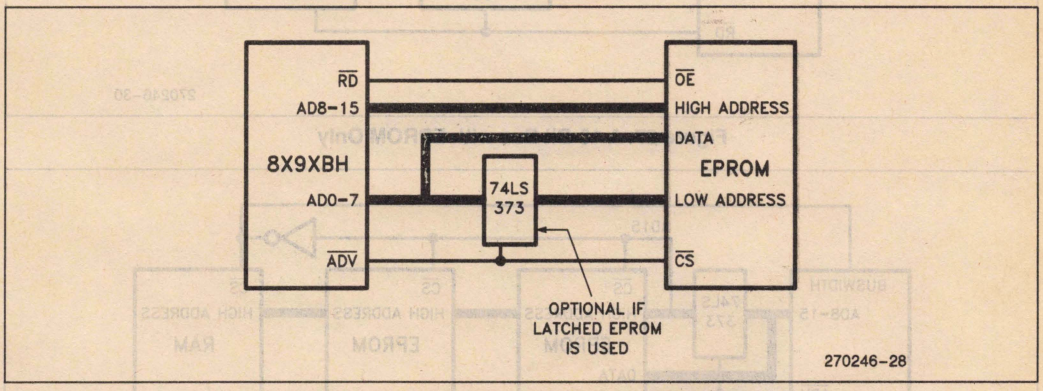


Figure 25. An 8-Bit Bus with EPROM Only

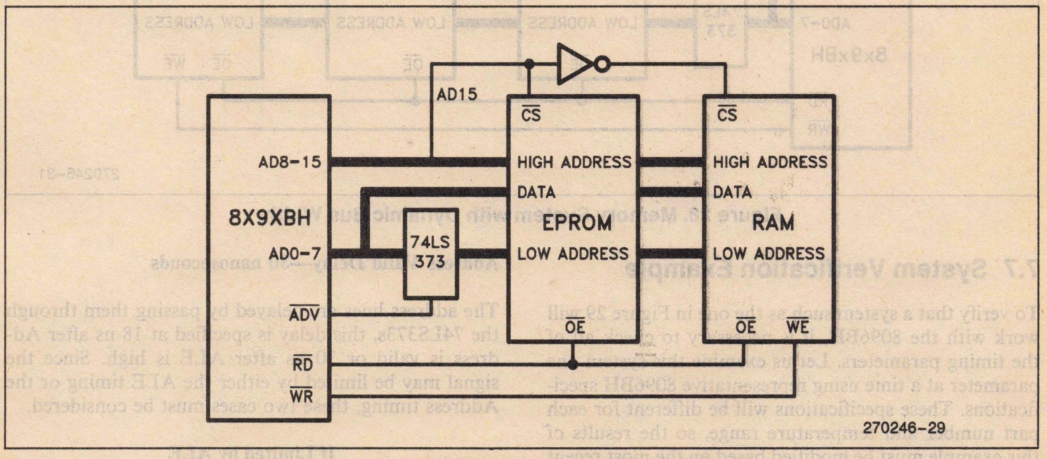


Figure 26. An 8-Bit Bus with EPROM and RAM

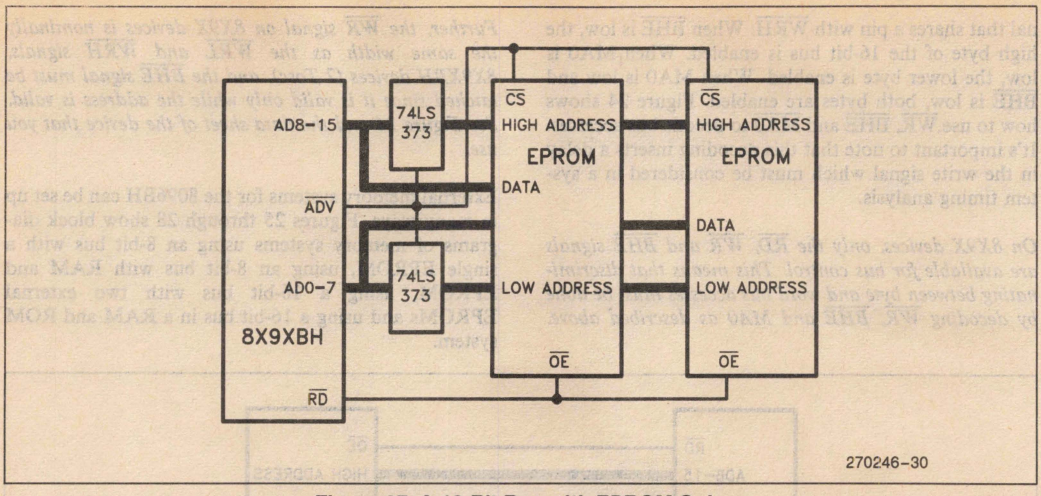


Figure 27. A 16-Bit Bus with EPROM Only

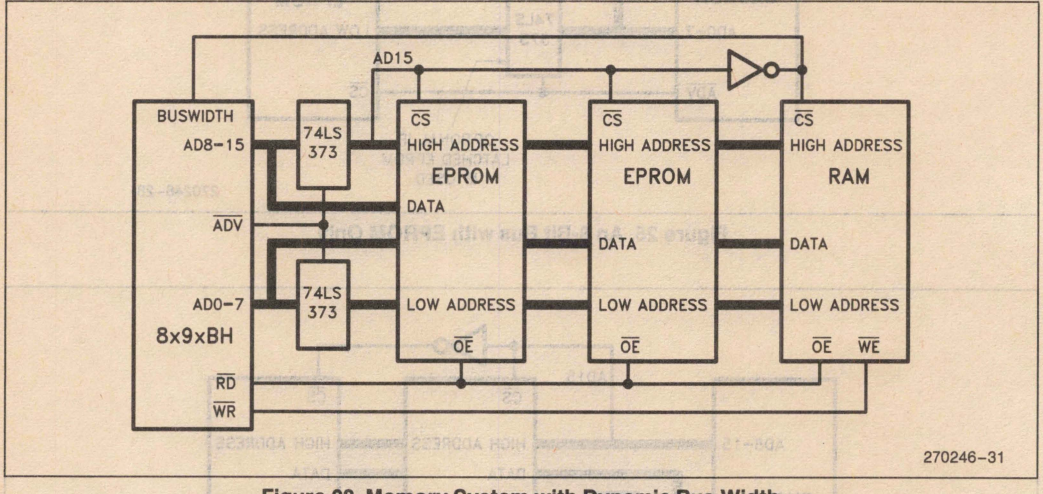


Figure 28. Memory System with Dynamic Bus Width

7.7 System Verification Example

To verify that a system such as the one in Figure 29 will work with the 8096BH, it is necessary to check all of the timing parameters. Let us examine this system one parameter at a time using representative 8096BH specifications. These specifications will be different for each part number and temperature range, so the results of this example must be modified based on the most recent data sheet for the specific part to be used.

The timings of signals that the processor and memory use are affected by the latch and buffer circuitry. The timings of the signal provided by the processor are delayed by various amounts of time. Similarly, the signals coming back from the memory are also delayed. The calculations involved in verifying this system follow:

Address Valid Delay—30 nanoseconds

The address lines are delayed by passing them through the 74LS373s, this delay is specified at 18 ns after Address is valid or 30 ns after ALE is high. Since the signal may be limited by either the ALE timing or the Address timing, these two cases must be considered.

If Limited by ALE

Minimum ALE pulse width = Tosc − 25 (TLHLL)

Minimum Addr set-up to ALE falling = Tosc − 25 (TAVLL)

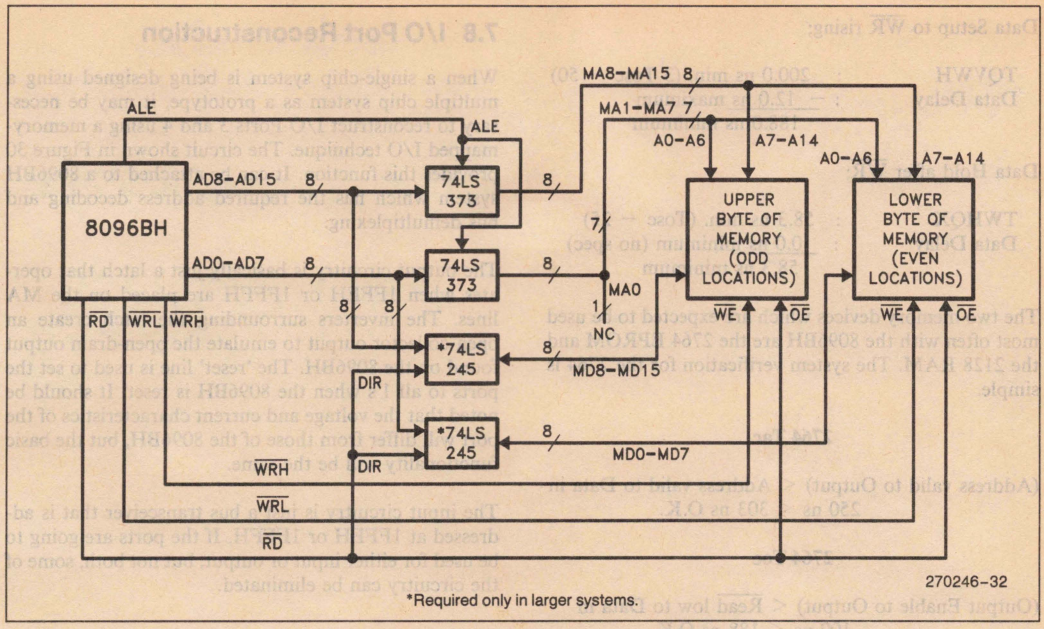


Figure 29. RAM/ROM Memory System

Therefore, in the worst case, ALE would occur 0 ns before Address valid.

Total delay from 8096BH Address stable to MA (Memory Address) stable would be:

ALE delay from address	− 0
74LS373 clock to output	30
	30 nanoseconds

If Limited by Address Valid

74LS373 Data Valid to Data Output = 18 nanoseconds

In the worst case, the delay in Address valid is controled by ALE and has a value of 30 nanoseconds.

Delay of Data Transfer to/from Processor—12 nanoseconds

The $\overline{RD}$ low to Data valid specification (TRLDV) is 3 Tosc − 50, (200 ns at 12 MHz). The 74LS245 is enabled by $\overline{RD}$ and has a delay of 40 ns from enable. The enable delay is clearly not a problem.

The 74LS245 is enabled for write, except during a read, so there is no enable delay to consider for write operations.

The Data In to Data Out delay of the 74LS245 is 12 ns.

CHARACTERISTICS OF A 12 MHz 8096BH SYSTEM WITH LATCHES

Required by system:

Address valid to Data in;

TAVDV	:	345.6 ns max. (5 Tosc − 70)
Address Delay	:	− 30.0 ns maximum
Data Delay	:	− 12.0 ns maximum
		303.6 ns maximum

Read low to Data in:

TRLDV	:	200.0 ns max. (3 Tosc − 50)
Address Delay	:	− 00.0 ns maximum
Data Delay	:	− 12.0 ns maximum
		188.0 ns maximum

Provided by system:

Address valid to Control;

TLLRL	:	63.3 ns min. (Tosc − 20)
TAVLL	:	158.3 ns min. (Tosc − 25)
Address Delay	:	− 30.0 ns maximum
$\overline{WR}$ Delay	:	− 00.0 ns minimum
		91.6 ns minimum

Write Pulse Width;

THLHH	:	146.6 ns min. (2 Tosc − 20)
		146.6 ns minimum

Data Setup to $\overline{WR}$ rising;

TQVWH	:	200.0 ns min. (3 Tosc − 50)
Data Delay	:	− 12.0 ns maximum
		188.0 ns minimum

Data Hold after $\overline{WR}$;

TWHQX	:	58.3 ns min. (Tosc − 25)
Data Delay	:	0.0 ns minimum (no spec)
		58.3 ns minimum

The two memory devices which are expected to be used most often with the 8096BH are the 2764 EPROM and the 2128 RAM. The system verification for the 2764 is simple.

2764 Tac

(Address valid to Output) < Address valid to Data in
250 ns < 303 ns O.K.

2764 Toe

(Output Enable to Output) < $\overline{Read}$ low to Data in
100 ns < 188 ns O.K.

These calculations assume no address decoder delays and no delays on the $\overline{RD}$ (OE) line. If there are delays in these signals the delays must be added to the 2764's timing.

The read calculations for the 2128 are similar to those for the 2764.

2128-20 Tac < Address valid to Data in
200 ns < 303 ns O.K.

2128-20 Toe < $\overline{Read}$ low to Data in
65 ns < 188 ns O.K.

The write calculation are a little more involved, but still straight-forward.

2128 Twp (Write Pulse) < Write Pulse Width
100 ns < 146 ns O.K.

2128 Tds (Data Setup) < Data Setup to $\overline{WR}$ rising
65 ns < 188 ns O.K.

2128 Tdh (Data Hold) < Data Hold after $\overline{WR}$
0 ns < 58 ns

All of the above calculations have been done assuming that no components are in the circuit except for those shown in Figure 29. If additional components are added, as may be needed for address decoding or memory bank switching, the calculations must be updated to reflect the actual circuit.

7.8 I/O Port Reconstruction

When a single-chip system is being designed using a multiple chip system as a prototype, it may be necessary to reconstruct I/O Ports 3 and 4 using a memory-mapped I/O technique. The circuit shown in Figure 30 provides this function. It can be attached to a 8096BH system which has the required address decoding and bus demultiplexing.

The output circuitry is basically just a latch that operates when 1FFEH or 1FFFH are placed on the MA lines. The inverters surrounding the latch create an open-collector output to emulate the open-drain output found on the 8096BH. The 'reset' line is used to set the ports to all 1's when the 8096BH is reset. It should be noted that the voltage and current characteristics of the port will differ from those of the 8096BH, but the basic functionality will be the same.

The input circuitry is just a bus transceiver that is addressed at 1FFEH or 1FFFH. If the ports are going to be used for either input or output, but not both, some of the circuitry can be eliminated.

8.0 NOISE PROTECTION TIPS

Designing controllers differs from designing other computer equipment in the area of noise protection. A microcontroller circuit under the hood of a car, in a photocopier, CRT terminal, or a high speed printer is subject to many types of electrical noise. Noise can get to the processor directly through the power supply, or it can be induced onto the board by electromagnetic fields. It is also possible for the PC board to find itself in the path of electrostatic discharges. Glitches and noise on the PC board can cause the processor to act unpredictably, usually by changing either the memory locations or the program counter.

There are both hardware and software solutions to noise problems, but the best solution is good design practice and a few ounces of prevention. The 8096BH has a Watchdog Timer which will reset the part if it fails to execute the software properly. The software should be set up to take advantage of this feature.

It is also recommended that unused areas of code be filled with NOPs and periodic jumps to an error routine or RST (reset chip) instructions. This is particularly important in the code around lookup tables, since if lookup tables are executed all sorts of bad things can happen. Wherever space allows, each table should be surrounded by 7 NOPs (the longest 8096BH instruction has 7 bytes) and a RST or jump to error routine instruction. This will help to ensure a speedy recovery should the processor have a glitch in the program flow.

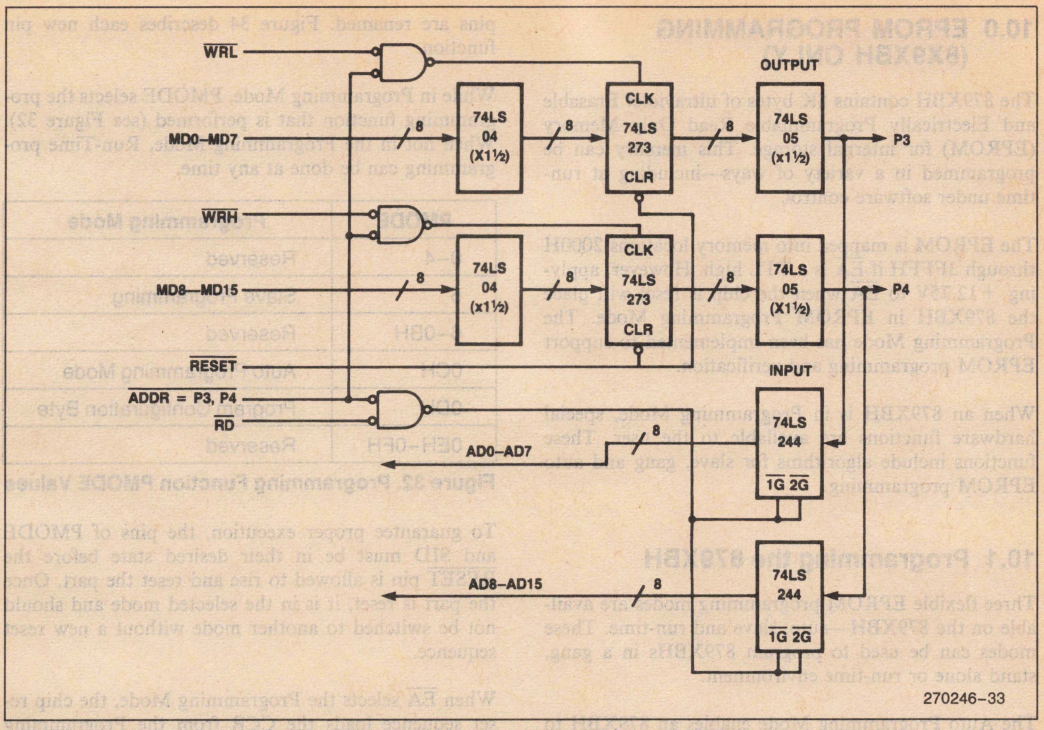


Figure 30. I/O Port Reconstruction

Many hardware solutions exist for keeping PC board noise to a minimum. Ground planes, gridded ground and VCC structures, bypass capacitors, transient absorbers and power busses with built-in capacitors can all be of great help. It is much easier to design a board with these features than to try to retrofit them later. Proper PC board layout is probably the single most important and, unfortunately, least understood aspect of project design. Minimizing loop areas and inductance, as well as providing clean grounds are very important. More information on protecting against noise can be found in the Application Note AP-125, "Designing Microcontroller Systems for Noisy Environments".

9.0 PACKAGING

The MCS-96 family of products is offered in many versions. They are available in 48-pin or 68-pin packages, with or without on-chip ROM/EPROM and with or without an A/D converter. A summary of the available options is shown in Figure 31.

The 48-pin versions are available in ceramic and plastic 48-pin Dual-In-Line package (DIP). The ceramic versions have part numbers with the prefix "C". The plastic versions have the prefix "P".

The 68-pin versions are available in a ceramic pin grid array (PGA), a plastic leaded chip carrier (PLCC) and a Type B leadless chip carrier (LCC). PGA devices have part numbers with the prefix "C". PLCC devices have the prefix "N". LCC devices have the prefix "R".

Specifications for the various members of the MCS-96 family are contained in the next chapter.

	ROMless		With ROM		With EPROM	
	68-pin	48-pin	68-pin	48-pin	68-pin	48-pin
Without A to D	8096		8396		8796	
With A to D	8097	8095	8397	8395	8797	8795

Figure 31. The MCS®-96 Family of Products

10.0 EPROM PROGRAMMING (8X9XBH ONLY)

The 879XBH contains 8K bytes of ultraviolet Erasable and Electrically Programmable Read Only Memory (EPROM) for internal storage. This memory can be programmed in a variety of ways—including at run-time under software control.

The EPROM is mapped into memory locations 2000H through 3FFFH if $\overline{EA}$ is a TTL high. However, applying +12.75V to $\overline{EA}$ when the chip is reset will place the 879XBH in EPROM Programming Mode. The Programming Mode has been implemented to support EPROM programming and verification.

When an 879XBH is in Programming Mode, special hardware functions are available to the user. These functions include algorithms for slave, gang and auto EPROM programming.

10.1 Programming the 879XBH

Three flexible EPROM programming modes are available on the 879XBH—auto, slave and run-time. These modes can be used to program 879XBHs in a gang, stand alone or run-time environment.

The Auto Programming Mode enables an 879XBH to program itself, and up to 15 other 879XBHs, with the 8K bytes of code beginning at address 4000H on its external bus. The Slave Mode provides a standard interface that enables any number of 879XBHs to be programmed by a master device such as an EPROM programmer. The Run-Time Mode allows individual EPROM locations to be programmed at run-time under complete software control.

In the Programming Mode, some I/O pins have been renamed. These new pin functions are used to determine the programming function that is performed, provide programming ALEs, provide slave ID numbers and pass error information. Figure 33 shows how the pins are renamed. Figure 34 describes each new pin function.

While in Programming Mode, PMODE selects the programming function that is performed (see Figure 32). When not in the Programming Mode, Run-Time programming can be done at any time.

PMODE	Programming Mode
0–4	Reserved
5	Slave Programming
6–0BH	Reserved
0CH	Auto Programming Mode
0DH	Program Configuration Byte
0EH–0FH	Reserved

Figure 32. Programming Function PMODE Values

To guarantee proper execution, the pins of PMODE and SID must be in their desired state before the $\overline{RESET}$ pin is allowed to rise and reset the part. Once the part is reset, it is in the selected mode and should not be switched to another mode without a new reset sequence.

When $\overline{EA}$ selects the Programming Mode, the chip reset sequence loads the CCR from the Programming Chip Configuration Byte (PCCB). This is a separate EPROM location that is not mapped under normal operation. PCCB is only important when programming in the Auto Programming Mode. In this mode, the 879XBH that is being programmed gets the data to be programmed from external memory over the system bus. Therefore, PCCR must correctly correspond to the memory system in the programming setup, which is not necessarily the memory organization of the application.

The following sections describe 879XBH programming in each programming mode.

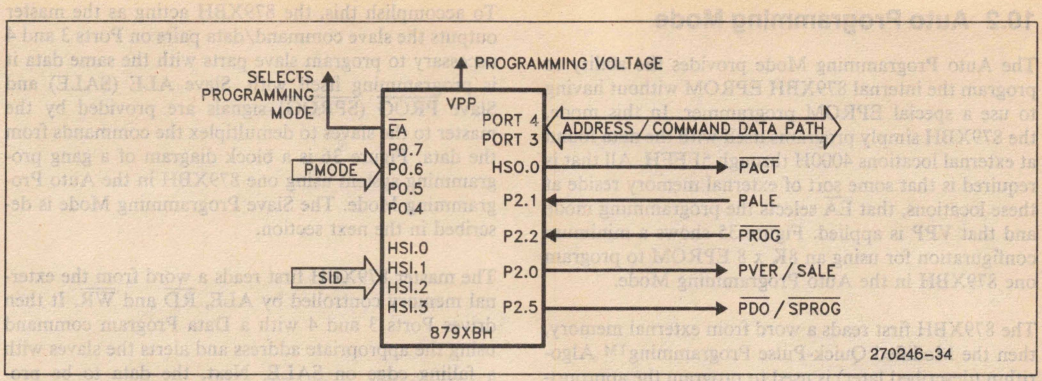


Figure 33. Programming Mode Pin Function

Name	Function
PMODE	**PROGRAMMING MODE SELECT:** Determines the EPROM programming algorithm that is performed. PMODE is sampled after a chip reset and should be static while the part is operating.
SID	**SLAVE ID NUMBER:** Used to assign each slave pin of Port 3 or 4 to use for passing programming verification acknowledgement. For example, if gang programming in the Slave Programming Mode, the slave with SID = 0001 will use Port 3.1 to signal correct or incorrect program verification.
PALE	**PROGRAMMING ALE INPUT**: Accepted by an 879XBH that is in the Slave Programming Mode. Used to indicate that Ports 3 and 4 contain a command/ address.
$\overline{\text{PROG}}$	**PROGRAMMING PULSE:** Accepted by 879XBH that is in the Slave Programming Mode. Used to indicate that Ports 3 and 4 contain the data to be programmed. A falling edge on $\overline{\text{PROG}}$ signifies data valid and starts the programming cycle. A rising edge on $\overline{\text{PROG}}$ will halt programming in the slaves.
$\overline{\text{PACT}}$	**PROGRAMMING ACTIVE:** Used in the Auto-Programming Mode to indicate when programming activity is complete.
PVER	**PROGRAM VERIFIED:** A signal outut after a programming operation by parts in the Slave Programming Mode.
$\overline{\text{PDO}}$	**PROGRAMMING DURATION OVERFLOWED:** A signal output by parts in the Slave Programming Mode. Used to signify that the $\overline{\text{PROG}}$ pulse applied for a programming operation was longer than allowed.
SALE	**SLAVE ALE:** Output signal from an 879XBH in the Auto Programming Mode. A falling edge on SALE indicates that Ports 3 and 4 contain valid address/command information for slave 879XBHs that may be attached to the master.
$\overline{\text{SPROG}}$	**SLAVE PROGRAMMING PULSE:** Output from an 879XBH in the Auto Programming Mode. A falling edge on $\overline{\text{SPROG}}$ indictates that Ports 3 and 4 contain valid data for programming into slave 879XBHs that may be attached to the master.
PORTS 3 and 4	**ADDRESS/COMMAND/DATA BUS:** Used to pass commands, addresses and data to and from slave mode 879XBHs. Used by chips in the Auto Programming Mode to pass command, addresses and data to slaves. Also used in the Auto Programming Mode as a regular system bus to access external memory. Each line should be pulled up to VCC through a resistor.

Figure 34. Programming Mode Pin Definitions

10.2 Auto Programming Mode

The Auto Programming Mode provides the ability to program the internal 879XBH EPROM without having to use a special EPROM programmer. In this mode, the 879XBH simply programs itself with the data found at external locations 4000H through 5FFFH. All that is required is that some sort of external memory reside at these locations, that $\overline{EA}$ selects the programming mode and that VPP is applied. Figure 35 shows a minimum configuration for using an 8K x 8 EPROM to program one 879XBH in the Auto Programming Mode.

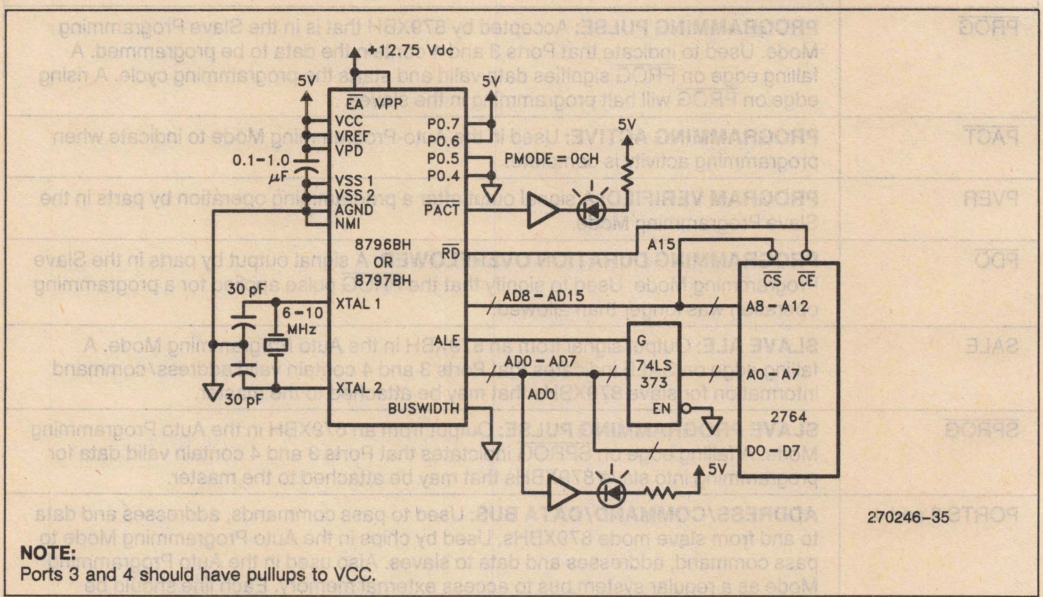
The 879XBH first reads a word from external memory, then the Modified Quick-Pulse Programming™ Algorithm (described later) is used to program the appropriate EPROM location. Since the erased state of a byte is 0FFH, the Auto Programming Mode will skip locations where the data to be programmed is 0FFH. When all 8K has been programmed, $\overline{PACT}$ goes high and the part outputs a 0 on Port 3.0 if it programmed correctly and a 1 if it failed.

10.2.1 GANG PROGRAMMING WITH THE AUTO PROGRAMMING MODE

An 879XBH in the Auto Programming Mode can also be used as a programmer for up to 15 other 879XBHs that are configured in the Slave Programming Mode. To accomplish this, the 879XBH acting as the master outputs the slave command/data pairs on Ports 3 and 4 necessary to program slave parts with the same data it is programming itself with. Slave ALE (SALE) and Slave $\overline{PROG}$ ($\overline{SPROG}$) signals are provided by the master to the slaves to demultiplex the commands from the data. Figure 36 is a block diagram of a gang programming system using one 879XBH in the Auto Programming Mode. The Slave Programming Mode is described in the next section.

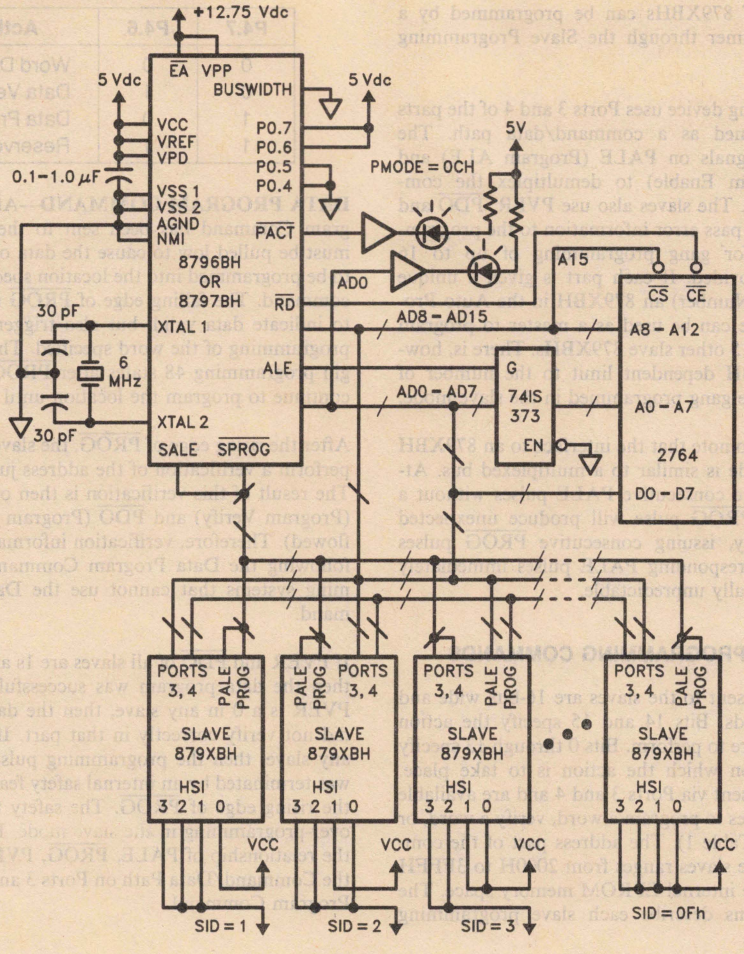
The master 879XBH first reads a word from the external memory controlled by ALE, $\overline{RD}$ and $\overline{WR}$. It then drives Ports 3 and 4 with a Data Program command using the appropriate address and alerts the slaves with a falling edge on SALE. Next, the data to be programmed is driven onto Ports 3 and 4 and slave programming begins with a falling edge on $\overline{SPROG}$. At the same time, the master begins to program its own EPROM location with the data read in. Intel's Modified Quick-Pulse Programming™ Algorithm is used, with Data Verify commands being given to the slaves after each programming pulse.

When programming is complete $\overline{PACT}$ goes high and Ports 3 and 4 are driven with all 1s if all parts programmed correctly. Individual bits of Port 3 and 4 will be driven to 0 if the slave with that bit number as an SID did not program correctly. The 879XBH used as the master assigns itself an SID of 0.



NOTE:
Ports 3 and 4 should have pullups to VCC.

Figure 35. The Auto Programming Mode



NOTE:
EA and VPP on slaves must be at +12.75 Vdc. Each slave's PMODE must equal 05H. Ports 3 and 4 should have pullups to VCC. Minimum configuration connections must also be made for slaves. A 10 MHz clock is recommended for the slaves.

Figure 36. Gang Programming with the Auto Programming Mode

10.3 Slave Programming Mode

Any number of 879XBHs can be programmed by a master programmer through the Slave Programming Mode.

The programming device uses Ports 3 and 4 of the parts being programmed as a command/data path. The slaves accept signals on PALE (Program ALE) and $\overline{\text{PROG}}$ (Program Enable) to demultiplex the commands and data. The slaves also use PVER, $\overline{\text{PDO}}$ and Ports 3 and 4 to pass error information to the programmer. Support for gang programming of up to 16 879XBHs is provided. If each part is given a unique SID (Slave ID Number) an 879XBH in the Auto Programming Mode can be used as a master to program itself and up to 15 other slave 879XBHs. There is, however, no 879XBH dependent limit to the number of parts that can be gang programmed in the slave mode.

It is important to note that the interface to an 879XBH in the slave mode is similar to a multiplexed bus. Attempting to issue consecutive PALE pulses without a corresponding $\overline{\text{PROG}}$ pulse will produce unexpected results. Similarly, issuing consecutive $\overline{\text{PROG}}$ pulses without the corresponding PALE pulses immediately preceding is equally unpredictable.

10.3.1 SLAVE PROGRAMMING COMMANDS

The commands sent to the slaves are 16-bits wide and contain two fields. Bits 14 and 15 specify the action that the slaves are to perform. Bits 0 through 13 specify the address upon which the action is to take place. Commands are sent via Ports 3 and 4 and are available to cause the slaves to program a word, verify a word, or dump a word (Table 1). The address part of the command sent to the slaves ranges from 2000H to 3FFFH and refers to the internal EPROM memory space. The following sections describe each slave programming mode command.

Table 1. Slave Programming Mode Commands

P4.7	P4.6	Action
0	0	Word Dump
0	1	Data Verify
1	0	Data Program
1	1	Reserved

DATA PROGRAM COMMAND—After a Data Program Command has been sent to the slaves, $\overline{\text{PROG}}$ must be pulled low to cause the data on Ports 3 and 4 to be programmed into the location specified during the command. The falling edge of $\overline{\text{PROG}}$ is not only used to indicate data valid, but also triggers the hardware programming of the word specified. The slaves will begin programming 48 states after $\overline{\text{PROG}}$ falls, and will continue to program the location until $\overline{\text{PROG}}$ rises.

After the rising edge of $\overline{\text{PROG}}$, the slaves automatically perform a verification of the address just programmed. The result of this verification is then output on PVER (Program Verify) and $\overline{\text{PDO}}$ (Program Duration Overflowed). Therefore, verification information is available following the Data Program Command for programming systems that cannot use the Data Verify command.

If PVER and $\overline{\text{PDO}}$ of all slaves are 1s after $\overline{\text{PROG}}$ rises then the data program was successful everywhere. If PVER is a 0 in any slave, then the data programmed did not verify correctly in that part. If $\overline{\text{PDO}}$ is a 0 in any slave, then the programming pulse in those parts was terminated by an internal safety feature rather than the rising edge of $\overline{\text{PROG}}$. The safety feature prevents over-programming in the slave mode. Figure 37 shows the relationship of PALE, $\overline{\text{PROG}}$, PVER and $\overline{\text{PDO}}$ to the Command/Data Path on Ports 3 and 4 for the Data Program Command.

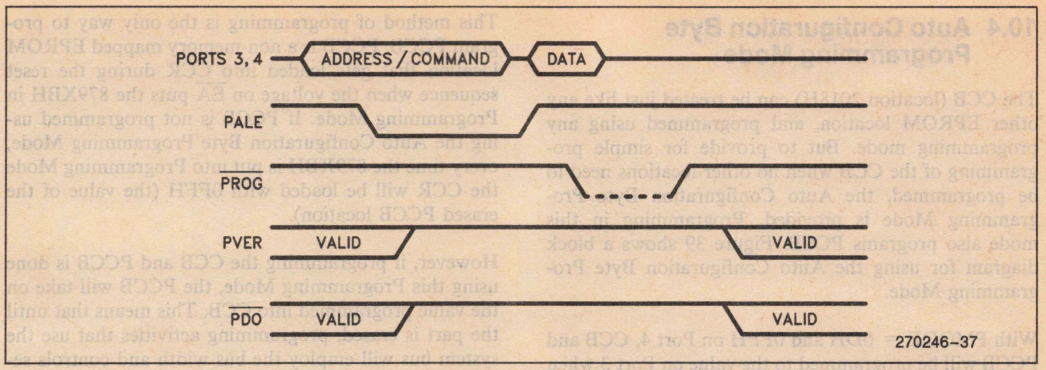


Figure 37. Data Program Signals in Slave Programming Mode

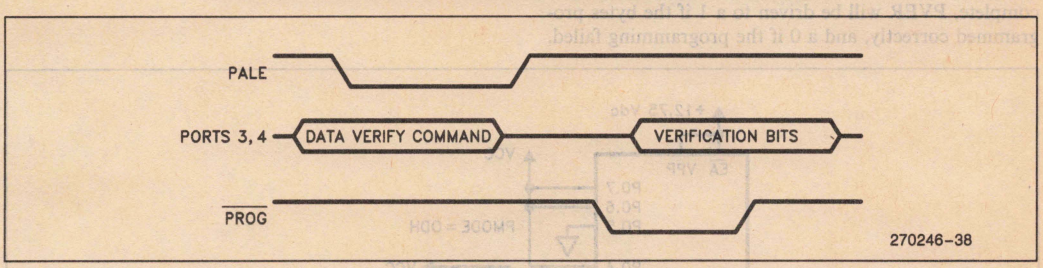


Figure 38. Data Verify Command Signals

DATA VERIFY COMMAND—When the Data Verify Command is sent, the slaves respond by driving one bit of Port 3 and 4 to indicate correct or incorrect verification of the previous Data Program. A 1 indicates correct verification, while a 0 indicates incorrect verification. The SID (Slave ID Number) of each slave determines which bit of the command/data path is driven. $\overline{\text{PROG}}$ from the programmer governs when the slaves drive the bus. Figure 38 shows the relationship of Ports 3 and 4 to PALE and $\overline{\text{PROG}}$.

This command is always preceded by a Data Program Command in a programming system with as many as 16 slaves. However, a Data Verify Command does not have to follow every Data Program Command.

WORD DUMP COMMAND—When the Word Dump Command is issued, the 879XBH being programmed adds 2000H to the address field of the command and places the value found at the new address on Ports 3 and 4. For example, sending the command #0100H to a slave will result in the slave placing the word found at location 2100H on Ports 3 and 4. $\overline{\text{PROG}}$ from the programmer governs when the slave drives the bus. The signals are the same as shown in Figure 22.

Note that this command will work only when just one slave is attached to the bus, and that there is no restriction on commands that precede or follow a Word Dump Command.

10.3.2 GANG PROGRAMMING WITH THE SLAVE PROGRAMMING MODE

Gang programming of 879XBHs can be done using the Slave Programming Mode. There is no 879XBH based limit on the number of chips that may be hooked to the same Port 3/Port 4 data path for gang programming.

If more than 16 chips are being gang programmed, the PVER and $\overline{\text{PDO}}$ outputs of each chip could be used for verification. The master programmer could issue a data program command then either watch every chip's error signals, or AND all the signals together to get a system PVER and $\overline{\text{PDO}}$.

If 16 or fewer 879XBHs are to be gang programmed at once, a more flexible form of verification is available. By giving each chip being programmed a unique SID, the master programmer could then issue a data verify command after the data program command. When a verify command is seen by the slaves, each will drive one pin of Port 3 or 4 with a 1 if the programming verified correctly or a 0 if programming failed. The SID is used by each slave to determine which Port 3, 4 bit it is assigned. An 879XBH in the Auto Programming Mode could be the master programmer if 15 or fewer slaves need to be programmed (see Gang Programming with the Auto Programming Mode).

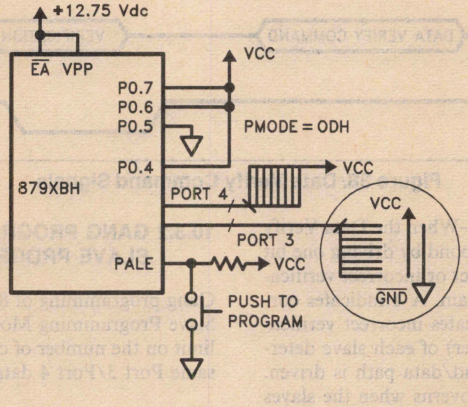
10.4 Auto Configuration Byte Programming Mode

The CCB (location 2018H) can be treated just like any other EPROM location, and programmed using any programming mode. But to provide for simple programming of the CCB when no other locations need to be programmed, the Auto Configuration Byte Programming Mode is provided. Programming in this mode also programs PCCB. Figure 39 shows a block diagram for using the Auto Configuration Byte Programming Mode.

With PMODE = 0DH and 0FFH on Port 4, CCB and PCCB will be programmed to the value on Port 3 when a logic 0 is placed on PALE. After programming is complete, PVER will be driven to a 1 if the bytes programmed correctly, and a 0 if the programming failed.

This method of programming is the only way to program PCCB. PCCB is a non-memory mapped EPROM location that gets loaded into CCR during the reset sequence when the voltage on $\overline{EA}$ puts the 879XBH in Programming Mode. If PCCB is not programmed using the Auto Configuration Byte Programming Mode, every time the 879XBH is put into Programming Mode the CCR will be loaded with 0FFH (the value of the erased PCCB location).

However, if programming the CCB and PCCB is done using this Programming Mode, the PCCB will take on the value programmed into CCB. This means that until the part is erased, programming activities that use the system bus will employ the bus width and controls selected by the user's CCB.



NOTES:
1. Tie Port 3 to the value desired to be programmed into CCB, and PCCB.
2. Make all necessary minimum connections for power, ground and clock.

Figure 39. The Auto CCR Programming Mode

10.5 Run-Time Programming

Run-Time Programming of the 879XBH is provided to allow the user complete flexibility in the ways in which the internal EPROM is programmed. That flexibility includes the ability to program just one byte or one word instead of the whole EPROM, and extends to the hardware necessary to program. The only additional requirement of a system is that a programming voltage is applied to VPP. Run-Time Programming is done with $\overline{EA}$ at TTL-high (normal operation—internal/external access).

To Run-Time program, the user writes a byte or word to the location to be programmed. Once this is done, the 879XBH will continue to program that location until another data read from or data write to the EPROM occurs. The user can therefore control the duration of the programming pulse to within a few mircoseconds. An intelligent algorithm should be implemented in software. It is recommended that the Modified Quick-Pulse Programming Algorithm be implemented.

After the programming of a location has started, care must be taken to ensure that no program fetches (or pre-fetches) occur from internal memory. This is of no concern if the program is executing from external memory. However, if the program is executing from internal memory when the write occurs, it will be necessary to use the built in "Jump to Self" located at 201AH.

"Jump to Self" is a two byte instruction in the Intel test ROM which can be CALLed after the user has started programming a location by writing to it. A software timer interrupt could then be used to escape from the "Jump to Self" when the proper programming pulse duration has elapsed. Figure 40 is an example of how to program an EPROM location while execution is entirely internal.

Upon entering the PROGRAM routine, the address and data are retrieved from the STACK and a Software Timer is set to expire one programming pulse later. The data is then written to the EPROM location and a CALL to location 201AH is made. Location 201AH is in Intel reserved test ROM, and contains the two byte opcode for a "Jump to Self". The minimum interrupt service routine would remove the 201AH return address from the STACK and return.

```
PROGRAM:

    POP  temp                          ;take parameters from the
                                        STACK
    POP  address_temp
    POP  data_temp
    PUSH temp

    PUSHF                              ;save current status
    LDB  int_mask , #enable_swt_only   ;enable only swt interrups
    LDB  HSO_COMMAND , #SWT0_ovf       ;load swt command to interrupt
    ADD  HSO_TIME,TIMER1, #program_pulse ;when program pulse time
                                        ;has elapsed
    EI
    ST   data-temp, [address_temp]
    CALL 201AH
    POPF
    RET

SWT_ISR:
    . . .

swt0_expired:
    POP  0
    RET
    . . .
```

Figure 40. Programming the EPROM from Internal Memory Execution

10.6 ROM/EPROM Program Lock

Protection mechanisms have been provided on the ROM and EPROM versions of the 8096BH to inhibit unauthorized accesses of internal program memory. However, there must always be a way to allow authorized program memory dumps for testing purposes. The following describes 839XBH, 879XBH program lock features and the mode provided for authorized memory dumps.

10.6.1 LOCK FEATURES

Write protection is provided for EPROM parts, while READ protection is provided for both ROM and EPROM parts.

Write protection is enabled by causing the LOC0 bit in the CCR to take the value 0. When WRITE protection is selected, the bus controller will cycle through the write sequence, but will not actually drive data to the EPROM and will not enable VPP to the EPROM. This protects the entire EPROM 2000H–3FFFH from inadvertant or unauthorized programming.

READ protection is selected by causing the LOC1 bit in the CCR to take the value 0. When READ protection is enabled, the bus controller will only perform a data read from the address range 2020H–3FFFH if the slave program counter is in the range 2000H–3FFFH. Note that since the slave PC can be many bytes ahead of the CPU program counter, an instruction that is located after address 3FFAH may not be allowed to access protected memory, even though the instruction is itself protected.

If the bus controller receives a request to perform a READ of protected memory, the READ sequence occurs with indeterminant data being returned to the CPU.

Other enhancements were also made to the 8096BH for program protection. For example, the value of $\overline{EA}$ is latched on reset so that the device cannot be switched from external to internal execution mode at run-time. In addition, if READ protection is selected, an NMI event will cause the device to switch to external only execution mode. Internal execution can only resume by resetting the chip.

10.6.2 AUTHORIZED ACCESS OF PROTECTED MEMORY

To provide a method of dumping the internal ROM/EPROM for testing purposes a "Security Key" mechanism and ROM dump mode have been implemented.

The security key is a 128 bit number, located in internal memory, that must be matched before a ROM dump will occur. The application code contains the security key starting at location 2020H.

The ROM dump mode is entered just like any programming mode ($\overline{EA}$ = 12.75V), except that a special PMODE strapping is used. The PMODE for ROM dump is 6H (0110B).

The ROM dump sequence begins with a security key verification. Users must place at external locations 4020H–402FH the same 16 byte key that resides inside the chip at locations 2020H–202FH. Before doing a ROM dump, the chip checks that the keys match.

After a successful key verification, the chip dumps data to external locations 1000H–11FFH and 4000H–5FFFH. Unspecified data appears at the low addresses.

Internal EPROM/ROM is dumped to 4000H–5FFFH, beginning with internal address 2000H.

If a security key verification is not successful, the chip will put itself into an endless loop of internal execution.

NOTE:
Substantial effort has been expended to provide an excellent program protection scheme. However, Intel cannot, and does not guarantee that the protection methods that we have devised will prevent unauthorized access.

10.7 Modified Quick-Pulse Programming™ Algorithm

The Modified Quick-Pulse Programming Algorithm calls for each EPROM location to receive 25 separate 100 μs (±5 μs) program cycles. Verification of correct programming is done after the 25 pulses. If the location verifies correctly, the next location is programmed. If the location fails to verify, the location has failed.

Once all locations are programmed and verified, the entire EPROM is again verified.

Programming of 879XBH parts is done with VPP = 12.75V ±0.25V and VCC = 5.0V ±0.5V.

10.8 Signature Word

The 8X9XBH contains a signature word at location 2070H. The word can be accessed in the slave mode by executing a word dump command.

Table 2. 8X9XBH Signature Words

Device	Signature Word
879XBH	896FH
839XBH	896EH
809XBH	Undefined

10.9 Erasing the 879XBH EPROM

Initially, and after each erasure, all bits of the 879XBH are in the "1" state. Data is introduced by selectively programming "0s" into the desired bit locations. Although only "0s" will be programmed, both "1s" and "0s" can be present in the data word. The only way to change a "0" to a "1" is by ultraviolet light erasure.

The erasure characteristics of the 879XBH are such that erasure begins to occur upon exposure to light with wavelengths shorter than approximately 4000 Angstroms (Å). It should be noted that sunlight and certain types of fluorescent lamps have wavelengths in the 3000–4000 Å range. Constant exposure to room level fluorescent lighting could erase the typical 879XBH in approximately 3 years, while it would take approximately 1 week to cause erasure when exposed to direct sunlight. If the 879XBH is to be exposed to light for extended periods of time, opaque labels must be placed over the EPROM's window to prevent unintentional erasure.

The recommended erasure procedure for the 879XBH is exposure to shortwave ultraviolet light which has a wavelength of 2537Å. The integrated dose (i.e., UV intensity × exposure time) for erasure should be a minimum of 15 Wsec/cm^2. The erasure time with this dosage is approximately 15 to 20 minutes using an ultraviolet lamp with a 12000 μW/cm^2 power rating. The 879XBH should be placed within 1 inch of the lamp tubes during erasure. The maximum integrated dose an 879XBH can be exposed to without damage is 7258 Wsec/cm^2 (1 week @ 12000 μW/cm^2). Exposure of the 879XBH to high intensity UV light for long periods may cause permanent damage.

10.7 Modified Quick-Pulse Programming™ Algorithm

The Modified Quick-Pulse Programming Algorithm calls for each EPROM location to receive 25 separate 100 µs (±5 µs) program cycles. Verification of correct programming is done after the 25 pulses. If the location verifies correctly, the next location is programmed. If the location fails to verify, the location has failed.

Once all locations are programmed and verified, the entire EPROM is again verified.

Programming of a 27X2BH parts is done with VPP = 12.75V ±0.25V and VCC = 5.0V ±0.5V.

10.8 Signature Word

The 8X27X2BH contains a signature word at location 2070H. The word can be accessed in the slave mode by executing a word dump command.

Table 2. 8X27X2BH Signature Words

Device	Signature Word
8727X2BH	3979H
8327X2BH	3979EH
8027X2BH	Undefined

10.9 Erasing the 27X2BH EPROM

Initially, and after each erasure, all bits of the 27X2BH are in the "1" state. Data is introduced by selectively programming "0s" into the desired bit locations. Although only "0s" will be programmed, both "1s" and "0s" can be present in the data word. The only way to change a "0" to a "1" is by ultraviolet light erasure.

The erasure characteristics of the 27X2BH are such that erasure begins to occur upon exposure to light with wavelengths shorter than approximately 4000 Angstroms (Å). It should be noted that sunlight and certain types of fluorescent lamps have wavelengths in the 3000–4000 Å range. Constant exposure to room level fluorescent lighting could erase the typical 27X2BH in approximately 3 years, while it would take approximately 1 week to cause erasure when exposed to direct sunlight. If the 27X2BH is to be exposed to light for extended periods of time, opaque labels must be placed over the EPROM's window to prevent unintentional erasure.

The recommended erasure procedure for the 27X2BH is exposure to ultraviolet light which has a wavelength of 2537Å. The integrated dose (i.e., UV intensity × exposure time) for erasure should be a minimum of 15 Wsec/cm². The erasure time with this dosage is approximately 15 to 20 minutes using an ultraviolet lamp with a 12000 µW/cm² power rating. The 27X2BH should be placed within 1 inch of the lamp tubes during erasure. The maximum integrated dose an 27X2BH can be exposed to without damage is 7258 Wsec/cm² (1 week @ 12000 µW/cm²). Exposure of the 27X2BH to high intensity UV light for long periods may cause permanent damage.

MCS®-96 Data Sheets

15

MCS®-96
809X-90, 839X-90

- **839X: an 809X with 8 Kbytes of On-Chip ROM**
- **High Speed Pulse I/O**
- **10-Bit A/D Converter**
- **6.5 μs 16 x 16 Multiply**
- **6.5 μs 32/16 Divide**
- **8 Interrupt Sources**
- **Pulse-Width Modulated Output**
- **232 Byte Register File**
- **Memory-to-Memory Architecture**
- **Full Duplex Serial Port**
- **Five 8-Bit I/O Ports**
- **Watchdog Timer**
- **Four 16-Bit Software Timers**

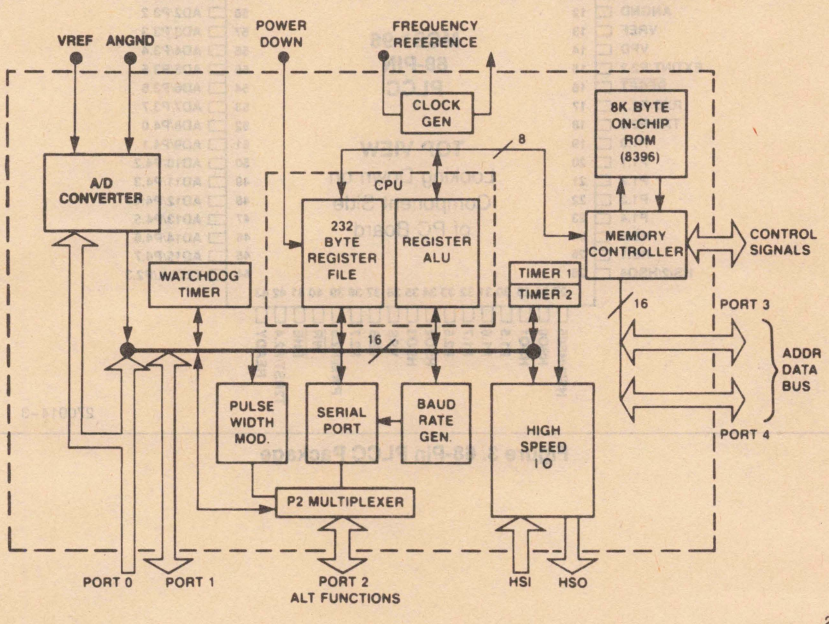
The MCS®-96 family of 16-bit microcontrollers consists of many members, all of which are designed for high-speed control functions. Members with the "−90" suffix are described in this data sheet.

The CPU supports bit, byte, and word operations. 32-bit double-words are supported for a subset of the instruction set. With a 12 MHz input frequency the 8096 can do a 16-bit addition in 1.0 μs and a 16 x 16-bit multiply or 32/16-bit divide in 6.5 μs. Instruction execution times average 1 to 2 μs in typical applications.

Four high-speed trigger inputs are provided to record the times at which external events occur. Six high-speed pulse generator outputs are provided to trigger external events at present times. The high-speed output unit can simultaneously perform timer functions. Up to four such 16-bit Software Timers can be in operation at once.

An on-chip A/D Converter converts up to 4 (in the 48-pin version) or 8 (in the 68-pin version) analog input channels to 10-bit digital values. This feature is only available on the 8095-90/8395-90 and 8097-90/8397-90.

Also provided on-chip are a serial port, a watchdog timer, and a pulse-width modulated output signal.



270014-1

Figure 1. Block Diagram

December 1986
Order Number: 270014-003

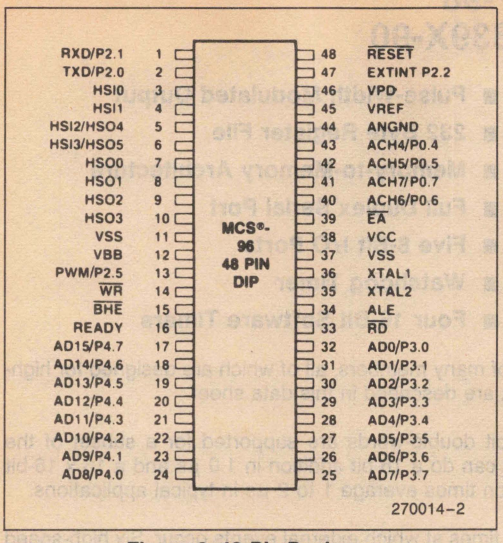


Figure 2. 48-Pin Package

Figure 1 shows a block diagram of the MCS-96 parts, generally referred to as the 8096. The 8096 is available in 48-pin and 68-pin packages, with and without A/D, and with and without on-chip ROM the MCS-96 numbering system is shown below:

Options		68-Pin	48-Pin
Digital I/O	ROMLESS	8096-90	8094-90
	ROM	8396-90	8394-90
Analog and Digital I/O	ROMLESS	8097-90	8095-90
	ROM	8397-90	8395-90

Figures 2, 3 and 4 show the pinouts for the 48- and 68-pin packages. The 48-pin version is offered in a Dual-In-Line package while the 68-pin version comes in a Plastic Leaded Chip Carrier and a Pin Grid Array.

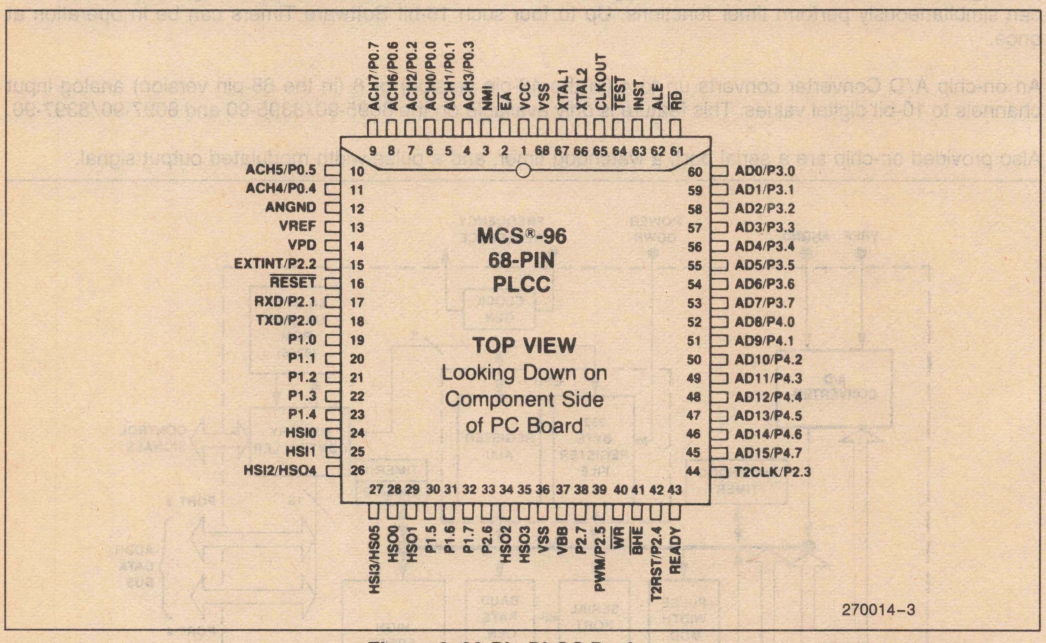


Figure 3. 68-Pin PLCC Package

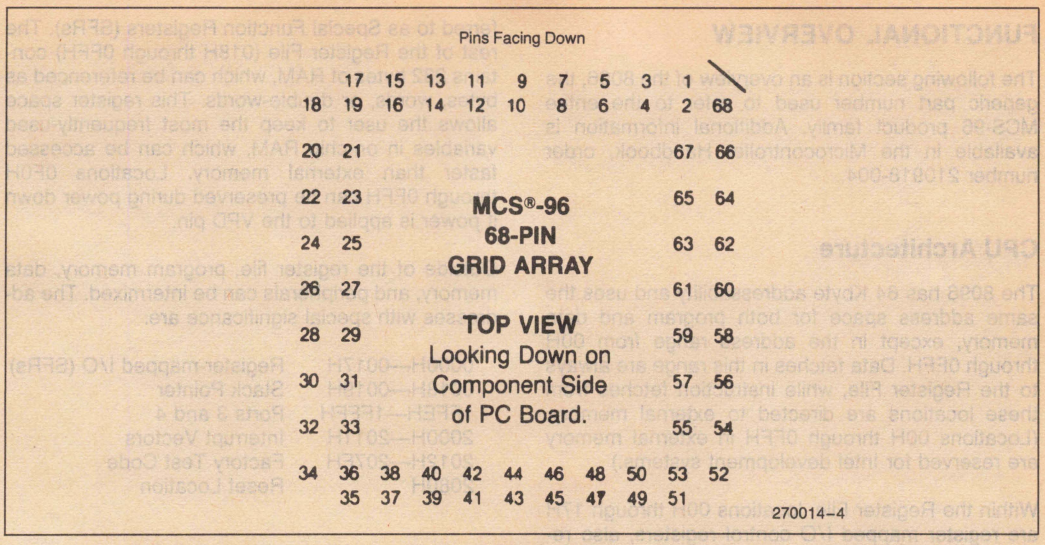


Figure 4. Pin Grid Array

PGA	PLCC	Description	PGA	PLCC	Description	PGA	PLCC	Description
1	9	ACH7/P0.7	24	54	AD6/P3.6	47	31	P1.6
2	8	ACH6/P0.6	25	53	AD7/P3.7	48	30	P1.5
3	7	ACH2/P0.2	26	52	AD8/P4.0	49	29	HSO.1
4	6	ACH0/P0.0	27	51	AD9/P4.1	50	28	HSO.0
5	5	ACH1/P0.1	28	50	AD10/P4.2	51	27	HSO.5/HSI.3
6	4	ACH3/P0.3	29	49	AD11/P4.3	52	26	HSO.4/HSI.2
7	3	NMI	30	48	AD12/P4.4	53	25	HSI.1
8	2	$\overline{EA}$	31	47	AD13/P4.5	54	24	HSI.0
9	1	VCC	32	46	AD14/P4.6	55	23	P1.4
10	68	VSS	33	45	AD15/P4.7	56	22	P1.3
11	67	XTAL1	34	44	T2CLK/P2.3	57	21	P1.2
12	66	XTAL2	35	43	READY	58	20	P1.1
13	65	CLKOUT	36	42	T2RST/P2.4	59	19	P1.0
14	64	$\overline{TEST}$	37	41	$\overline{BHE}$	60	18	TXD/P2.0
15	63	INST	38	40	$\overline{WR}$	61	17	RXD/P2.1
16	62	ALE	39	39	PWM/P2.5	62	16	$\overline{RESET}$
17	61	$\overline{RD}$	40	38	P2.7	63	15	EXTINT/P2.2
18	60	AD0/P3.0	41	37	VBB	64	14	VPD
19	59	AD1/P3.1	42	36	VSS	65	13	VREF
20	58	AD1/P3.2	43	35	HSO.3	66	12	ANGND
21	57	AD3/P3.3	44	34	HSO.2	67	11	ACH4/P0.4
22	56	AD4/P3.4	45	33	P2.6	68	10	ACH5/P0.5
23	55	AD5/P3.5	46	32	P1.7			

FUNCTIONAL OVERVIEW

The following section is an overview of the 8096, the generic part number used to refer to the entire MCS-96 product family. Additional information is available in the Microcontroller Handbook, order number 210918-004.

CPU Architecture

The 8096 has 64 Kbyte addressability and uses the same address space for both program and data memory, except in the address range from 00H through 0FFH. Data fetches in this range are always to the Register File, while instruction fetches from these locations are directed to external memory. (Locations 00H through 0FFH in external memory are reserved for Intel development systems.)

Within the Register File, locations 00H through 17H are register mapped I/O control registers, also referred to as Special Function Registers (SFRs). The rest of the Register File (018H through 0FFH) contains 232 bytes of RAM, which can be referenced as bytes, words, or double-words. This register space allows the user to keep the most frequently-used variables in on-chip RAM, which can be accessed faster than external memory. Locations 0F0H through 0FFH can be preserved during power down if power is applied to the VPD pin.

Outside of the register file, program memory, data memory, and peripherals can be intermixed. The addresses with special significance are:

0000H—0017H	Register-mapped I/O (SFRs)
0018H—0019H	Stack Pointer
1FFEH—1FFFH	Ports 3 and 4
2000H—2011H	Interrupt Vectors
2012H—207FH	Factory Test Code
2080H	Reset Location

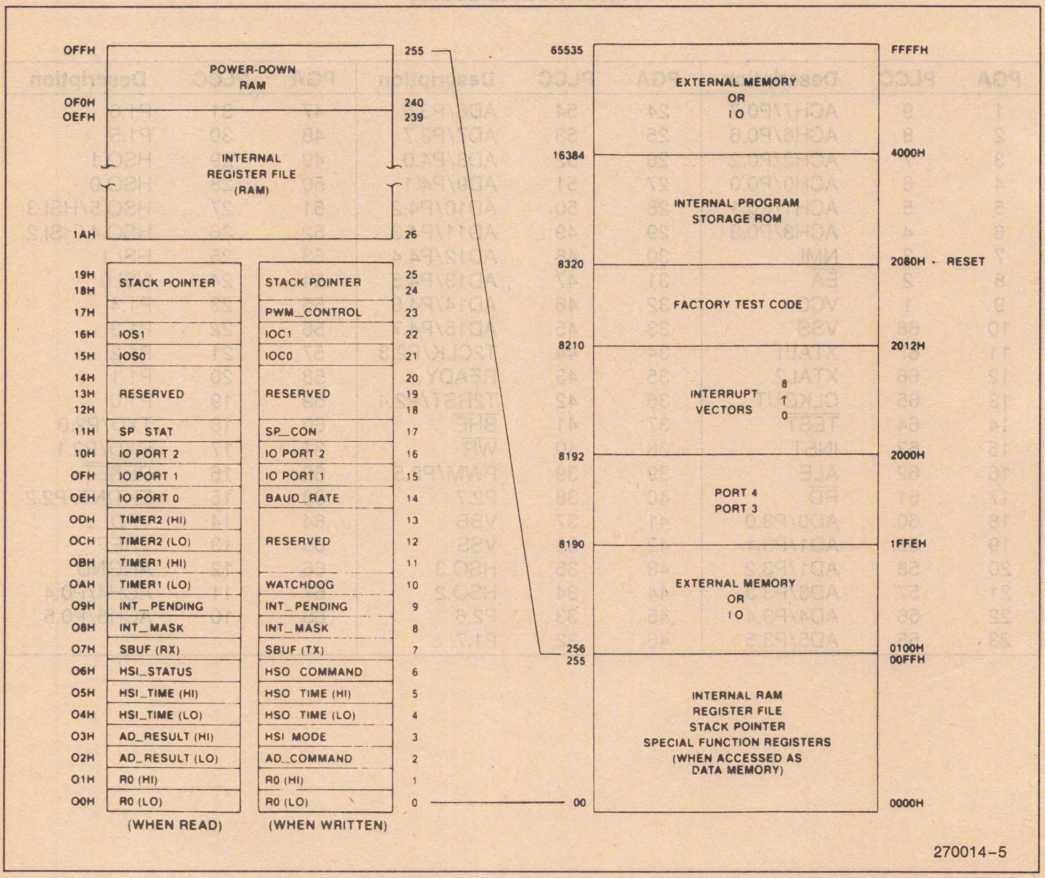


Figure 5. Memory Map

The 839x carries 8 Kbytes of on-chip ROM, occupying addresses 2000H through 3FFFH. Instruction or data fetches from these addresses access the on-chip ROM if the $\overline{EA}$ pin is externally held at a logical 1. If the $\overline{EA}$ pin is at a logical 0 these addresses access off-chip memory.

A memory map for the MCS-96 product family is shown in Figure 5.

The RALU (Register/ALU) section consists of a 17-bit ALU, the Program Status Word, the Program Counter, and several temporary registers. A key feature of the 8096 is that it does not use an accumulator. Rather, it operates directly on any register in the Register File. Being able to operate directly on data in the Register File without having to move it into and out of an accumulator results in a significant improvement in execution speed.

In addition to the normal arithmetic and logical functions, the MCS-96 instruction set provides the following special features:

6.5 μs Multiply and Divide
Multiple Shift Instructions

3 Operand Instructions
Normalize Instruction
Software Reset Instruction

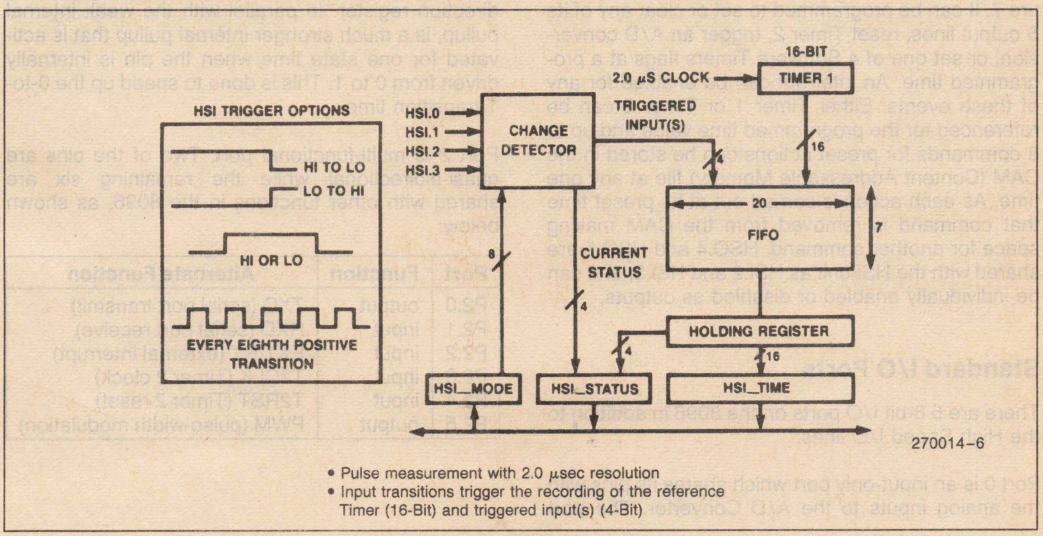
All operations on the 8096 take place in a set number of "State Times." The 8096 uses a three-phase internal clock, so each state time is 3 oscillator periods. With a 12 MHz clock, each state time requires 0.25 microseconds.

High Speed I/O Unit (HSIO)

The HSIO unit consists of the High Speed Input Unit (HSI), the High Speed Output Unit (HSO), one counter and one timer. "High Speed" denotes that the units can perform functions related to the timers without CPU intervention. The HSI records times when events occur and the HSO triggers events at preprogrammed times.

All actions within the HSIO unit are synchronized to the timers. The two 16-bit timer/counter registers in the HSIO unit are cleared on chip reset and can be programmed to generate an interrupt on overflow. The Timer 1 register is automatically incremented every 8 state times (every 2.0 microseconds, with a 12 MHz clock). The Timer 2 register can be programmed to count transitions on either the T2CLK pin or HSI.1 pin. It is incremented on both positive and negative edges of the selected input line. In addition to being cleared by reset, Timer 2 can also be cleared in software or by signals from input pins T2RST or HS1.0. Neither of these timers is required for the watchdog timer or the serial port.

The High Speed Input (HSI) unit can detect transitions on any of its 4 input lines. When one occurs it records the time (from Timer 1) and which input lines



- Pulse measurement with 2.0 μsec resolution
- Input transitions trigger the recording of the reference Timer (16-Bit) and triggered input(s) (4-Bit)

Figure 6. High Speed Input Unit

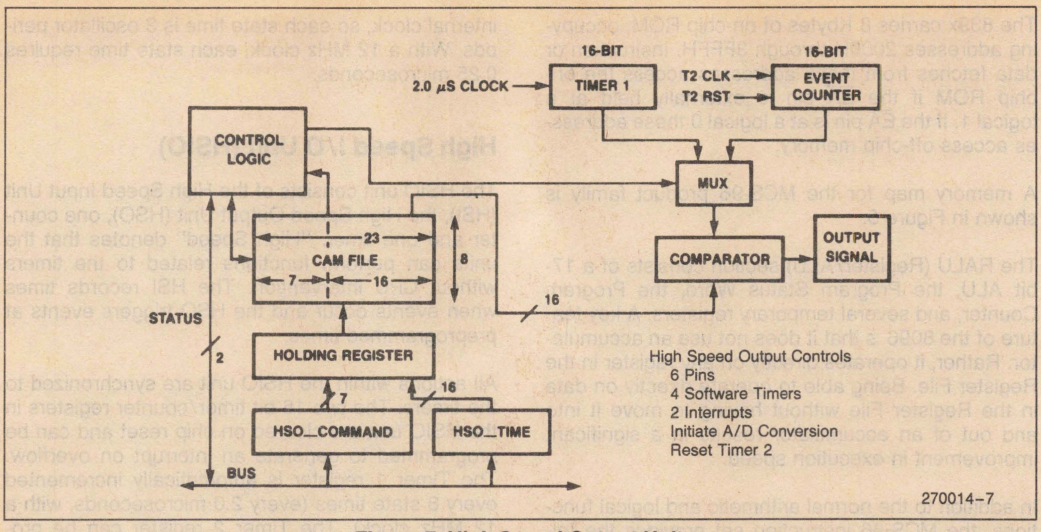


Figure 7. High Speed Output Unit

made the transition. This information is recorded with 2 microsecond resolution and stored in an 8-level FIFO. The unit can be programmed to look for four types of events, as shown in Figure 6. It can activate the HSI Data Available interrupt either when the Holding Registers is loaded or the 6th FIFO entry has been made. Each input line can be individually enabled or disabled to the HSI unit by software.

The High Speed Output (HSO) unit is shown in Figure 7. It can be programmed to set or clear any of its 6 output lines, reset Timer 2, trigger an A/D conversion, or set one of 4 Software Timers flags at a programmed time. An interrupt can be enabled for any of these events. Either Timer 1 or Timer 2 can be referenced for the programmed time value and up to 8 commands for preset actions can be stored in the CAM (Content Addressable Memory) file at any one time. As each action is carried out at its preset time that command is removed from the CAM making space for another command. HSO.4 and HSO.5 are shared with the HSI unit as HSI.2 and HSI.3, and can be individually enabled or disabled as outputs.

Standard I/O Ports

There are 5 8-bit I/O ports on the 8096 in addition to the High Speed I/O lines.

Port 0 is an input-only port which shares its pins with the analog inputs to the A/D Converter. The port can be read digitally and/or, by writing to the A/D Command Register, one of the lines can be selected as the input to the A/D Converter.

Port 1 is a quasi-bidirectional I/O port. "Quasi-bidirectional" means the port pin has a weak internal pullup that is always active and an internal pulldown which can either be on (to output a 0) or off (to output a 1). This configuration allows the pin to be used as either an input or an output without using a data direction register. In parallel with the weak internal pullup, is a much stronger internal pullup that is activated for one state time when the pin is internally driven from 0 to 1. This is done to speed up the 0-to-1 transition time.

Port 2 is multi-functional port. Two of the pins are quasi-bidirectional while the remaining six are shared with other functions in the 8096, as shown below:

Port	Function	Alternate Function
P2.0	output	TXD (serial port transmit)
P2.1	input	RXD (serial port receive)
P2.2	input	EXTINT (external interrupt)
P2.3	input	T2CLK (Timer 2 clock)
P2.4	input	T2RST (Timer 2 reset)
P2.5	output	PWM (pulse-width modulation)

Ports 3 and 4 are bi-directional I/O ports with open drain outputs. These pins are also used as the multiplexed address/data bus when accessing external memory, in which case they have strong internal pullups. The internal pullups are only used during external memory read or write cycles when the pins are outputting address or data bits. At any other time, the internal pullups are disabled.

Serial Port

The serial port is compatible with the MCS®-51 family (8051, 8031 etc.) serial port. It is full duplex, and receive-buffered. There are 3 asynchronous modes and 1 synchronous mode of operation for the serial port. The asynchronous modes allow for 8 or 9 bits of data with even parity optionally inserted for one of the data bits. Selective interrupts based on the 9th data bit are available to support interprocessor communication.

Baud rates in all modes are determined by an independent 16-bit on-chip baud rate generator. Either the XTAL 1 pin or the T2CLK pin can be used as the input to the baud rate generator. The maximum baud rate in the asynchronous mode is 187.5 KBaud.

Pulse Width Modulator (PWM)

The PWM output shares a pin with port bit P2.5. When the PWM output is selected, this pin outputs a pulse train having a fixed period of 256 state times, and a programmable width of 0 to 255 state times. The width is programmed by loading the desired value, in state times, to the PWM Control Register.

A/D Converter

The analog-to-digital converter is a 10-bit, successive approximation converter. It has a fixed conversion time of 168 state times, (42 microseconds with a 12 MHz clock). The analog input must be in the range of 0 to VREF (normally, VREF = 5V). This input can be selected from 8 analog input lines, which connect to the same pins as Port 0. A conversion can be initiated either by setting a control bit in the A/D Command register, or by programming the HSO unit to trigger the conversion at some specified time.

Interrupts

The 8096 has 20 interrupt sources which vector through 8 locations. A 0-to-1 transition from any of the sources sets a corresponding bit in the Interrupt Pending register. The content of the Interrupt Mask register determines if a pending interrupt will be serviced or not. if it is to be serviced, the CPU pushes the current program counter onto the stack and reloads it with the vector corresponding to the desired interrupt. The interrupt vectors are located in addresses 2000H through 2011H, as shown in Figure 8.

Source	Vector Location (High Byte)	Vector Location (Low Byte)	Priority
Software	2011H	2010H	Not Applicable
Extint	200FH	200EH	7 (Highest)
Serial Port	200DH	200CH	6
Software Timers	200BH	200AH	5
HSI.0	2009H	2008H	4
High Speed Outputs	2007H	2006H	3
HSI Data Available	2005H	2004H	2
A/D Conversion Complete	2003H	2002H	1
Timer Overflow	2001H	2000H	0 (Lowest)

Figure 8. Interrupt Vectors

At the end of the terminal routine the RET instruction pops the program counter from the stack and execution continues where it left off. It is not necessary to store and replace registers during interrupt routines as each routine can be set up to use a different section of the register file. This feature of the architecture provides for very fast context switching.

While the 8096 has a single priority level in the sense that any interrupt may be itself be interrupted, a priority structure exists for resolving simultaneously pending interrupts, as indicated in Figure 8. Since the interrupt pending and interrupt mask registers can be manipulated in software, it is possible to dynamically alter the interrupt priorities to suit the users' software.

Watchdog Timer

The watchdog timer is a 16-bit counter which, once started, is incremented every state time. After 16 milliseconds, if not cleared, it will overflow, pulling down the $\overline{\text{RESET}}$ pin for two state times, causing the system to be reinitialized. This feature is provided as a means of graceful recovery from a software upset. The counter must be cleared by the software before it overflows, or else the system assumes an upset has occurred and activates $\overline{\text{RESET}}$.

PIN DESCRIPTION

VCC

Main supply voltage (5V).

VSS

Digital circuit ground (0V).

VPD

RAM standby supply voltage (5V). This voltage must be present during normal operation. In a Power Down condition (i.e., VCC drops to zero), if $\overline{\text{RESET}}$ is activated before VCC drops below spec and VPD continues to be held within spec, the top 16 bytes in the Register File will retain their contents. $\overline{\text{RESET}}$ must be held low during the Power Down and should not be brought high until VCC is within spec and the oscillator has stablized.

VREF

Reference voltage for the A/D converter (5V). VREF is also the supply voltage to the analog portion of the A/D converter and the logic used to read Port 0.

ANGND

Reference ground for the A/D converter. Should be held at nominally the same potential as VSS.

VBB

Substrate voltage from the on-chip back-bias generator. This pin should be connected to ANGND through a 0.01 μf capacitor (and not connected to anything else).

XTAL1

Input of the oscillator inverter and of the internal clock generator.

XTAL2

Output of the oscillator inverter.

CLKOUT

Output of the internal clock generator. The frequency of CLKOUT is 1/3 the oscillator frequency. It has a 33% duty cycle.

$\overline{\text{RESET}}$

Reset input to the chip. Input low for at least 2 state times to reset the chip. The subsequent low-to-high transition re-synchronizes CLKOUT and commences a 10-state-time sequence in which the PSW is cleared and a jump to address 2080H is executed. Input high for normal operation. $\overline{\text{RESET}}$ has an internal pullup.

TEST

Input low enables a factory test mode. The user should tie this pin to VCC for normal operation.

NMI

A positive transition clears the watchdog timer, and causes a vector to external memory location 0000H. External memory from 00H through 0FFH is reserved for Intel development systems.

INST

Output high during an external memory read indicates the read is an instruction fetch.

$\overline{\text{EA}}$

Input for memory select (External Access). $\overline{\text{EA}}$ = 1 causes memory accesses to locations 2000H through 3FFFH to be directed to on-chip ROM. $\overline{\text{EA}}$ = 0 causes accesses to these locations to be directed to off-chip memory. $\overline{\text{EA}}$ has an internal pulldown, so it goes to 0 unless driven to 1.

ALE

Address Latch Enable output. ALE is activated only during external memory accesses. It is used to latch the address from the multiplexed address/data bus.

$\overline{\text{RD}}$

Read signal output to external memory. $\overline{\text{RD}}$ is activated only during external memory reads.

$\overline{\text{WR}}$

Write signal output to external memory. $\overline{\text{WR}}$ is activated only during external memory writes.

$\overline{BHE}$

Bus High Enable signal output to external memory. $\overline{BHE}$ = 0 selects the bank of memory that is connected to the high byte of the data bus. A0 = 0 selects the bank of memory that is connected to the low byte of the data bust. Thus accesses to a 16-bit wide memory can be to the low byte only (A0 = 0, $\overline{BHE}$ = 1), to the high byte only (A0 = 1, $\overline{BHE}$ = 0), or to both bytes (A0 = 0, $\overline{BHE}$ = 0). $\overline{BHE}$ is activated only when required during accesses to external memory. $\overline{BHE}$ can be ignored during read operations.

READY

The READY input is used to lengthen external memory bus cycles, for interfacing to slow or dynamic memory, or for bus sharing. If the pin is high CPU operation continues in a normal manner. If the pin is low prior to the rising edge of CLKOUT, the Memory Controller goes into a wait mode until the next negative transition in CLKOUT occurs with READY high. The bus cycle can be lengthened by up to 1 μs. When the external memory bus is not being used, READY has no effect. READY has a weak internal pullup, so it goes to 1 unless externally pulled low.

HSI

Inputs to High Speed Input Unit. Four HSI pins are available: HSI.0, HSI.1, HSI.2, and HSI.3. Two of them (HSI.2 and HSI.3) are shared with the HSO Unit.

HSO

Outputs from High Speed Output Unit. Six HSO pins are available: HSO.0, HSO.1, HSO.2, HSO.3, HSO.4, and HSO.5. Two of them (HSO.4 and HSO.5) are shared with the HSI Unit.

Port 0

8-bit high impedance input-only port. These pins can be used as digital inputs and/or as analog inputs to the on-chip A/D converter.

Port 1

8-bit quasi-bidirectional I/O port.

Port 2

8-bit multi-functional port. Six of its pins are shared with other functions in the 8096, the remaining 2 are quasi-bidirectional.

Ports 3 and 4

8-bit bi-directional I/O ports with open drain outputs. These pins are shared with the multiplexed address/data bus which has strong internal pullups.

INSTRUCTION SET

The 8096 instruction set makes use of six addressing modes as described below:

DIRECT—The operand is specified by an 8-bit address field in the instruction. The operand must be in the Register File or SFR space (locations 0000H through 00FFH).

IMMEDIATE—The operand itself follows the opcode in the instruction stream as immediate data. The immediate data can be either 8-bits or 16-bits as required by the opcode.

INDIRECT—An 8-bit address field in the instruction gives the address of a word register in the Register File which contains the 16-bit address of the operand. The operand can be anywhere in memory.

INDIRECT WITH AUTO-INCREMENT—Same as Indirect, except that, after the operand is referenced, the word register that contains the operand's address is incremented by 1 if the operand is a byte, or by 2 if the operand is a word.

INDEXED—The instruction contains an 8-bit address field and either an 8-bit or a 16-bit displacement field. The 8-bit address field gives the address of a word register in the Register File which contains a 16-bit base address. The 8- or 16-bit displacement field contains a signed displacement that will be added to the base address to produce the address of the operand. The operand can be anywhere in memory.

The 8096 contains a Zero Register at word address 0000H (and which contains 0000H). This register is available for performing comparisons and for use as a base register in indexed addressing. This effectively provides direct addressing to all 64K of memory.

In the 8096, the Stack Pointer is at word address 0018H in the Register File. If the 8-bit address field in an indexed instruction contains 18H, the Stack Pointer becomes the base register. This allows direct accessing of variables in the stack.

The following tables list the MCS-96 instructions, their opcodes, and execution times.

Instruction Summary

Mnemonic	Operands	Operation (Note 1)	Flags						Notes
			Z	N	C	V	VT	ST	
ADD/ADDB	2	D ← D + A	↕	↕	↕	↕	↑	—	
ADD/ADDB	3	D ← B + A	↕	↕	↕	↕	↑	—	
ADDC/ADDCB	2	D ← D + A + C	↓	↕	↕	↕	↑	—	
SUB/SUBB	2	D ← D − A	↕	↕	↕	↕	↑	—	
SUB/SUBB	3	D ← B − A	↕	↕	↕	↕	↑	—	
SUBC/SUBCB	2	D ← D − A + C − 1	↓	↕	↕	↕	↑	—	
CMP/CMPB	2	D − A	↕	↕	↕	↕	↑	—	
MUL/MULU	2	D, D + 2 ← D * A	—	—	—	—	—	?	2
MUL/MULU	3	D, D + 2 ← B * A	—	—	—	—	—	?	2
MULB/MULUB	2	D, D + 1 ← D * A	—	—	—	—	—	?	3
MULB/MULUB	3	D, D + 1 ← B * A	—	—	—	—	—	?	3
DIVU	2	D ← (D, D + 2)/A, D + 2 ← remainder	—	—	—	↕	↑	—	2
DIVUB	2	D ← (D, D + 1)/A, D + 1 ← remainder	—	—	—	↕	↑	—	3
DIV	2	D ← (D, D + 2)/A, D + 2 ← remainder	—	—	—	?	↑	—	2
DIVB	2	D ← (D, D + 1)/A, D + 1 ← remainder	—	—	—	?	↑	—	3
AND/ANDB	2	D ← D and A	↕	↕	0	0	—	—	
AND/ANDB	3	D ← B and A	↕	↕	0	0	—	—	
OR/ORB	2	D ← D or A	↕	↕	0	0	—	—	
XOR/XORB	2	D ← D (excl. or) A	↕	↕	0	0	—	—	
LD/LDB	2	D ← A	—	—	—	—	—	—	
ST/STB	2	A ← D	—	—	—	—	—	—	
LDBSE	2	D ← A; D + 1 ← SIGN(A)	—	—	—	—	—	—	3, 4
LDBZE	2	D ← A; D + 1 ← 0	—	—	—	—	—	—	3, 4
PUSH	1	SP ← SP − 2; (SP) ← A	—	—	—	—	—	—	
POP	1	A ← (SP); SP ← SP + 2	—	—	—	—	—	—	
PUSHF	0	SP ← SP − 2; (SP) ← PSW; PSW ← 0000H I ← 0	0	0	0	0	0	0	
POPF	0	PSW ← (SP); SP ← SP + 2; I ← ↕	↕	↕	↕	↕	↕	↕	
SJMP	1	PC ← PC + 11-bit offset	—	—	—	—	—	—	5
LJMP	1	PC ← PC + 16-bit offset	—	—	—	—	—	—	5
BR (indirect)	1	PC ← (A)	—	—	—	—	—	—	
SCALL	1	SP ← SP − 2; (SP) ← PC; PC ← PC + 11-bit offset	—	—	—	—	—	—	5
LCALL	1	SP ← SP − 2; (SP) ← PC; PC ← PC + 16-bit offset	—	—	—	—	—	—	5
RET	0	PC ← (SP); SP ← SP + 2	—	—	—	—	—	—	
J (conditional)	1	PC ← PC + 8-bit offset (if taken)	—	—	—	—	—	—	5
JC	1	Jump if C = 1	—	—	—	—	—	—	5
JNC	1	Jump if C = 0	—	—	—	—	—	—	5
JE	1	Jump if Z = 1	—	—	—	—	—	—	5

NOTES:
1. If the mnemonic ends in "B", a byte operation is performed, otherwise a word operation is done. Operands D, B, and A must conform to the alignment rules for the required operand type. D and B are locations in the register file; A can be located anywhere in memory.
2. D, D + 2 are consecutive WORDS in memory; D is DOUBLE-WORD aligned.
3. D, D + 1 are consecutive BYTES in memory; D is WORD aligned.
4. Changes a byte to a word.
5. Offset is a 2's complement number.

Instruction Summary (Continued)

Mnemonic	Operands	Operation (Note 1)	Flags						Notes
			Z	N	C	V	VT	ST	
JNE	1	Jump if Z = 0	—	—	—	—	—	—	5
JGE	1	Jump if N = 0	—	—	—	—	—	—	5
JLT	1	Jump if N = 1	—	—	—	—	—	—	5
JGT	1	Jump if N = 0 and Z = 0	—	—	—	—	—	—	5
JLE	1	Jump if N = 1 or Z = 1	—	—	—	—	—	—	5
JH	1	Jump if C = 1 and Z = 0	—	—	—	—	—	—	5
JNH	1	Jump if C = 0 or Z = 1	—	—	—	—	—	—	5
JV	1	Jump if V = 1	—	—	—	—	—	—	5
JNV	1	Jump if V = 0	—	—	—	—	—	—	5
JVT	1	Jump if VT = 1; Clear VT	—	—	—	—	0	—	5
JNVT	1	Jump if VT = 0; Clear VT	—	—	—	—	0	—	5
JST	1	Jump if ST = 1	—	—	—	—	—	—	5
JNST	1	Jump if ST = 0	—	—	—	—	—	—	5
JBS	3	Jump if Specified Bit = 1	—	—	—	—	—	—	5, 6
JBC	3	Jump if Specified Bit = 0	—	—	—	—	—	—	5, 6
DJNZ	1	D ← D − 1; if D ≠ 0 then PC ← PC + 8-bit offset	—	—	—	—	—	—	5
DEC/DECB	1	D ← D − 1	↙	↙	↙	↙	↑	—	
NEG/NEGB	1	D ← 0 − D	↙	↙	↙	↙	↑	—	
INC/INCB	1	D ← D + 1	↙	↙	↙	↙	↑	—	
EXT	1	D ← D; D + 2 ← Sign (D)	↙	↙	0	0	—	—	2
EXTB	1	D ← D; D + 1 ← Sign (D)	↙	↙	0	0	—	—	3
NOT/NOTB	1	D ← Logical Not (D)	↙	↙	0	0	—	—	
CLR/CLRB	1	D ← 0	1	0	0	0	—	—	
SHL/SHLB/SHLL	2	C ← msb — — — — — lsb ← 0	↙	?	↙	↙	↑	—	7
SHR/SHRB/SHRL	2	0 → msb — — — — — lsb → C	↙	?	↙	0	—	↙	7
SHRA/SHRAB/SHRAL	2	msb → msb — — — — — lsb → C	↙	↙	↙	0	—	↙	7
SETC	0	C ← 1	—	—	1	—	—	—	
CLRC	0	C ← 0	—	—	0	—	—	—	
CLRVT	0	VT ← 0	—	—	—	—	0	—	
RST	0	PC ← 2080H	0	0	0	0	0	0	8
DI	0	Disable All Interrupts (I ← 0)	—	—	—	—	—	—	
EI	0	Enable All Interrupts (I ← 1)	—	—	—	—	—	—	
NOP	0	PC ← PC + 1	—	—	—	—	—	—	
SKIP	0	PC ← PC + 2	—	—	—	—	—	—	
NORML	2	Left Shift Till msb = 1; D ← shift count	↙	?	0	—	—	—	7
TRAP	0	SP ← SP − 2; (SP) ← PC PC ← (2010H)	—	—	—	—	—	—	9

NOTES:

1. If the mnemonic ends in "B", a byte operation is performed, otherwise a word operation is done. Operands D, B, and A must conform to the alignment rules for the required operand type. D and B are locations in the register file; A can be located anywhere in memory.
5. Offset is a 2's complement number.
6. Specified bit is one of the 2048 bits in the register file.
7. The "L" (Long) suffix indicates double-word operation.
8. Initiates a Reset by pulling RESET low. Software should re-initialize all the necessary registers with code starting at 2080H.
9. The assembler will not accept this mnemonic.

MNEMONIC	OPERANDS	DIRECT			IMMEDIATE			INDIRECT* NORMAL			INDIRECT* AUTO-INC.		INDEXED* SHORT			INDEXED* LONG	
		OPCODE	BYTES	STATE TIMES	OPCODE	BYTES	STATE TIMES	OPCODE	BYTES	STATE① TIMES	BYTES	STATE① TIMES	OPCODE	BYTES	STATE① TIMES	BYTES	STATE① TIMES
ARITHMETIC INSTRUCTIONS																	
ADD	2	64	3	4	65	4	5	66	3	6/11	3	7/12	67	4	6/11	5	7/12
ADD	3	44	4	5	45	5	6	46	4	7/12	4	8/13	47	5	7/12	6	8/13
ADDB	2	74	3	4	75	3	4	76	3	6/11	3	7/12	77	4	6/11	5	7/12
ADDB	3	54	4	5	55	4	5	56	4	7/12	4	8/13	57	5	7/12	6	8/13
ADDC	2	A4	3	4	A5	4	5	A6	3	6/11	3	7/12	A7	4	6/11	5	7/12
ADDCB	2	B4	3	4	B5	3	4	B6	3	6/11	3	7/12	B7	4	6/11	5	7/12
SUB	2	68	3	4	69	4	5	6A	3	6/11	3	7/12	6B	4	6/11	5	7/12
SUB	3	48	4	5	49	5	6	4A	4	7/12	4	8/13	4B	5	7/12	6	8/13
SUBB	2	78	3	4	79	3	4	7A	3	6/11	3	7/12	7B	4	6/11	5	7/12
SUBB	3	58	4	5	59	4	5	5A	4	7/12	4	8/13	5B	5	7/12	6	8/13
SUBC	2	A8	3	4	A9	4	5	AA	3	6/11	3	7/12	AB	4	6/11	5	7/12
SUBCB	2	B8	3	4	B9	3	4	BA	3	6/11	3	7/12	BB	4	6/11	5	7/12
CMP	2	88	3	4	89	4	5	8A	3	6/11	3	7/12	8B	4	6/11	5	7/12
CMPB	2	98	3	4	99	3	4	9A	3	6/11	3	7/12	9B	4	6/11	5	7/12
MULU	2	6C	3	25	6D	4	26	6E	3	27/32	3	28/33	6F	4	27/32	5	28/33
MULU	3	4C	4	26	4D	5	27	4E	4	28/33	4	29/34	4F	5	28/33	6	29/34
MULUB	2	7C	3	17	7D	3	17	7E	3	19/24	3	20/25	7F	4	19/24	5	20/25
MULUB	3	5C	4	18	5D	4	18	5E	4	20/25	4	21/26	5F	5	20/25	6	21/26
MUL	2	②	4	29	②	5	30	②	4	31/36	4	32/37	②	5	31/36	6	32/37
MUL	3	②	5	30	②	6	31	②	5	32/37	5	33/38	②	6	32/37	7	33/38
MULB	2	②	4	21	②	4	21	②	4	23/28	4	24/29	②	5	23/28	6	24/29
MULB	3	②	5	22	②	5	22	②	5	24/29	5	25/30	②	6	24/29	7	25/30
DIVU	2	8C	3	25	8D	4	26	8E	3	28/32	3	29/33	8F	4	28/32	5	29/33
DIVUB	2	9C	3	17	9D	3	17	9E	3	20/24	3	21/25	9F	4	20/24	5	21/25
DIV	2	②	4	29	②	5	30	②	4	32/36	4	33/37	②	5	32/36	6	33/37
DIVB	2	②	4	21	②	4	21	②	4	24/28	4	25/29	②	5	24/28	6	25/29

270014-9

NOTES:

* Long indexed and Indirect + instructions have identical opcodes with Short indexed and Indirect modes, respectively. The second byte of instructions using any indirect or indexed addressing mode specifies the exact mode used. If the second byte is even, use Indirect or Short Indexed. If it is odd, use Indirect+ or Long Indexed. In all cases the second byte of the instruction always specifies an even (word) location for the address referenced.

1. Number of state times shown for internal/external operands.
2. The opcodes for signed multiply and divide are the opcodes for the unsigned functions with an "FE" appended as a prefix.
3. State times shown for 86-bit bus.

MNEMONIC	OPERANDS	DIRECT			IMMEDIATE			INDIRECT② NORMAL			INDIRECT② AUTO-INC.		INDEXED② SHORT			INDEXED② LONG	
		OPCODE	BYTES	STATE TIMES	OPCODE	BYTES	STATE TIMES	OPCODE	BYTES	STATE① TIMES	BYTES	STATE① TIMES	OPCODE	BYTES	STATE① TIMES	BYTES	STATE① TIMES
LOGICAL INSTRUCTIONS																	
AND	2	60	3	4	61	4	5	62	3	6/11	3	7/12	63	4	6/11	5	7/12
AND	3	40	4	5	41	5	6	42	4	7/12	4	8/13	43	5	7/12	6	8/13
ANDB	2	70	3	4	71	3	4	72	3	6/11	3	7/12	73	4	6/11	5	7/12
ANDB	3	50	4	5	51	4	5	52	4	7/12	4	8/13	53	5	7/12	6	8/13
OR	2	80	3	4	81	4	5	82	3	6/11	3	7/12	83	4	6/11	5	7/12
ORB	2	90	3	4	91	3	4	92	3	6/11	3	7/12	93	4	6/11	5	7/12
XOR	2	84	3	4	85	4	5	86	3	6/11	3	7/12	87	4	6/11	5	7/12
XORB	2	94	3	4	95	3	4	96	3	6/11	3	7/12	97	4	6/11	5	7/12
DATA TRANSFER INSTRUCTIONS																	
LD	2	A0	3	4	A1	4	5	A2	3	6/11	3	7/12	A3	4	6/11	5	7/12
LDB	2	B0	3	4	B1	3	4	B2	3	6/11	3	7/12	B3	4	6/11	5	7/12
ST	2	C0	3	4	—	—	—	C2	3	7/11	3	8/12	C3	4	7/11	5	8/12
STB	2	C4	3	4	—	—	—	C6	3	7/11	3	8/12	C7	4	7/11	5	8/12
LDBSE	2	BC	3	4	BD	3	4	BE	3	6/11	3	7/12	BF	4	6/11	5	7/12
LDBZE	2	AC	3	4	AD	3	4	AE	3	6/11	3	7/12	AF	4	6/11	5	7/12
STACK OPERATIONS (internal stack)																	
PUSH	1	C8	2	8	C9	3	8	CA	2	11/15	2	12/16	CB	3	11/15	4	12/16
POP	1	CC	2	12	—	—	—	CE	2	14/18	2	14/18	CF	3	14/18	4	14/18
PUSHF	0	F2	1	8													
POPF	0	F3	1	9													
STACK OPERATIONS (external stack)																	
PUSH	1	C8	2	12	C9	3	12	CA	2	15/19	2	16/20	CB	3	15/19	4	16/20
POP	1	CC	2	14	—	—	—	CE	2	16/20	2	16/20	CF	3	16/20	4	16/20
PUSHF	0	F2	1	12													
POPF	0	F3	1	13													

JUMPS AND CALLS							
MNEMONIC	OPCODE	BYTES	STATES	MNEMONIC	OPCODE	BYTES	STATES
LJMP	E7	3	8	LCALL	EF	3	13/16⑤
SJMP	20-27④	2	8	SCALL	28-2F④	2	13/16⑤
BR[ ]	E3	2	8	RET	F0	1	12/16⑤
				TRAP③	F7	1	21/24

270014-10

NOTES:
1. Number of state times shown for internal/external operands.
3. The assembler does not accept this mnemonic.
4. The least significant 3 bits of the opcode are concatenated with the following 8 bits to form an 11-bit, 2's complement, offset for the relative call or jump.
5. State times for stack located internal/external.

Conditional Jumps

All conditional jumps are 2 byte instructions. They require 8 state times if the jump is taken, 4 if it is not.

Mnemonic	Opcode	Mnemonic	Opcode	Mnemonic	Opcode	Mnemonic	Opcode
JC	DB	JE	DF	JGE	D6	JGT	D2
JNC	D3	JNE	D7	JLT	DE	JLE	DA
JH	D9	JV	DD	JVT	DC	JST	D8
JNH	D1	JNV	D5	JNVT	D4	JNST	D0

Jump on Bit Clear or Bit Set

These instructions are 3-byte instructions. They require 9 state times if the jump is taken, 5 if it is not.

	Bit Number							
Mnemonic	0	1	2	3	4	5	6	7
JBC	30	31	32	33	34	35	36	37
JBS	38	39	3A	3B	3C	3D	3E	3F

LOOP CONTROL

DJNZ	OPCODE E0;	3 BYTES;	5/9 STATE TIMES (NOT TAKEN/TAKEN)

Single Register Instructions

Mnemonic	Opcode	Bytes	States	Mnemonic	Opcode	Bytes	States
DEC	05	2	4	EXT	06	2	4
DECB	15	2	4	EXTB	16	2	4
NEG	03	2	4	NOT	02	2	4
NEGB	13	2	4	NOTB	12	2	4
INC	07	2	4	CLR	01	2	4
INCB	17	2	4	CLRB	11	2	4

Shift Instructions

Instr Mnemonic	Word		Instr Mnemonic	Byte		Instr Mnemonic	DBL WD		State Times
	OP	B		OP	B		OP	B	
SHL	09	3	SHLB	19	3	SHLL	0D	3	7 + 1 PER SHIFT(7)
SHR	08	3	SHRB	18	3	SHRL	0C	3	7 + 1 PER SHIFT(7)
SHRA	0A	3	SHRAB	1A	3	SHRAL	0E	3	7 + 1 PER SHIFT(7)

Special Control Instructions

Mnemonic	Opcode	Bytes	States	Mnemonic	Opcode	Bytes	States
SETC	F9	1	4	DI	FA	1	4
CLRC	F8	1	4	EI	FB	1	4
CLRVT	FC	1	4	NOP	FD	1	4
RST (6)	FF	1	166	SKIP	00	2	4

Normalize

Mnemonic	Opcode	Bytes	State Times
NORML	0F	3	11 + 1 PER SHIFT

NOTES:
6. This instruction takes 2 states to pull RST low, then holds it low for 2 states to initiate a reset. The reset takes 12 states, at which time the program restarts at location 2080H.
7. Execution will take at least 8 states, even for 0 shift.

FUNCTIONAL DEVIATIONS

Functional deviations from the 809x and 839x on the 809x-90 and 839x-90.

CPU Section

1. Indexed, 3 Word Multiply—The displacement portion of an indexed, three word multiply may not be in the range of 200H thru 17FFH inclusive.
2. Add or Subtract with carry—The zero flag is both set and cleared by these instructions.
3. EXT—This instruction never sets the N flag, and always sets the Z flag. The EXTB works correctly.
4. Read-Modify-Write on Interrupt Pending—A read-modify-write instruction on the interrupt pending register may cause interrupts that occur during execution of the instruction to be missed.
5. READY line—The READY line should not be brought low during the execution of an instruction that accesses HSI_TIME, SP_STAT or IOS1. It should also not be brought low for a data write during the instruction immediately preceding one of the above operations.

   The READY line also should not be brought low for more than two state times when using the EXT (extend word) instruction.
6. Signed Divide—The V and VT flags may indicate an overflow after a signed divide when no overflow has occurred.

HSI/HSO Section

1. HSI Timing—An event occurring within 16 state times of a prior event on the same HSI line may not be recorded. Additionally, an event occurring within 16 state times of a prior event on another HSI line may be recorded with a time tag one count earlier than expected. Events are defined as the condition the line is set to trigger on.
2. HSI Divide by 8 Mode—If an event on a pin set to look for every eighth transition occurs less than 16 state times after an event on any other pin, then the divide by 8 event will be recorded twice in the HSI FIFO. The time tag of the duplicate FIFO entry will be equal to that of the initial entry plus one.
3. HSO Interrupts—Software timer interrupts cannot be generated by the HSO commands that reset Timer 2 or start an A to D conversion.

Serial Port Section

1. Serial Port Flags—Reading SP_STAT may not clear the TI or RI flag if that flag was set within two state times prior to the read. In addition, the parity error bit (RPE/RB8) may not be correct if it is read within two state times after RI is set.
2. Serial Port Mode 0—The serial port is not tested in mode 0. The receive function in this mode does not word correctly.
3. Serial Port Baud Value—Loading the baud rate register with 8000H (maximum baud rate, internal clock) may cause an 11 millisecond delay (at Fosc = 12 MHz) before the port is properly initialized. After initialization the port works properly.

Standard I/O Section

1. Ports 3 and 4 (Internal Execution Mode Only)—To be used as outputs, Ports 3 and 4 each must be addressed as words but written to as bytes. To write to Port 3 use "ST temp, 1ffeh", where the low byte of "temp" contains the data for the port. To write to Port 4, use the DCB operator to generate the opcode sequence "0C3H, 001H, 0FFH, 01FH, (temp)", where the high byte of "temp" contains the data for the port. Ports 3 and 4 will not work as input ports.

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias . . . . 0°C to +70°C
Storage Temperature . . . . . −40°C to +150°C
Voltage from Any Pin to
V_{SS} or ANGND . . . . . −0.3V to +7.0V
Average Output Current from Any Pin . . . . . 10 mA
Power Dissipation . . . . . 1.5 Watts

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

NOTICE: Specifications contained within the following tables are subject to change.

OPERATING CONDITIONS

Symbol	Parameter	Min	Max	Units
T_A	Ambient Temperature Under Bias	0	+70	C
V_{CC}	Digital Supply Voltage	4.50	5.50	V
V_{REF}	Analog Supply Voltage	4.5	5.5	V
f_{OSC}	Oscillator Frequency	6.0	12	MHz
V_{PD}	Power-Down Supply Voltage	4.50	5.50	V

NOTE:
V_{BB} should be connected to ANGND through a 0.01 μF capacitor. ANGND and V_{SS} should be nominally at the same potential.

D.C. CHARACTERISTICS

Symbol	Parameter	Min	Max	Units	Test Conditions
V_{IL}	Input Low Voltage (Except $\overline{RESET}$)	−0.3	+0.8	V	
V_{IL1}	Input Low Voltage, $\overline{RESET}$	−0.3	+0.7	V	
V_{IH}	Input High Voltage (Except $\overline{RESET}$, NMI, XTAL1)	2.0	V_{CC} +0.5	V	
V_{IH1}	Input High Voltage, $\overline{RESET}$ Rising	2.4	V_{CC} +0.5	V	
V_{IH2}	Input High Voltage, $\overline{RESET}$ Falling	2.1	V_{CC} +0.5	V	
V_{IH3}	Input High Voltage, NMI, XTAL1	2.4	V_{CC} +0.5	V	
V_{OL}	Output Low Voltage		0.45	V	(Note 1)
V_{OH}	Output High Voltage	2.4		V	(Note 2)
I_{CC}	V_{CC} Supply Current		200	mA	All Outputs Disconnected
I_{PD}	VPD Supply Current		1	mA	Normal operation and Power-Down
I_{REF}	V_{REF} Supply Current		8	mA	
I_{LI}	Input Leakage Current to all pins of HSI, P3, P4, and to P2.1		±10	μA	V_{in} = 0 to V_{CC}
I_{LI1}	Input Leakage to Port 0		±3	μA	V_{IN} = 0 to V_{CC}
I_{IH}	Input High Current to $\overline{EA}$		100	μA	V_{IH} = 2.4V
I_{IL}	Input Low Current to all pins of P1, and to P2.6, P2.7		−100	μA	V_{IL} = 0.45V
I_{IL1}	Input Low Current to $\overline{RESET}$	0.3	−2	mA	V_{IL} = 0.45V
I_{IL2}	Input Low Current P2.2, P2.3, P2.4, READY		−50	μA	V_{IL} = 0.45V
C_S	Pin Capacitance (Any Pin to V_{SS})		10	pF	fTEST = 1.0 MHz

NOTES:
1. I_{OL} = 0.4 mA for all pins of P1, for P2.6 and P2.7, and for all pins of P3 and P4 when used as ports. I_{OL} = 2.0 mA for TXD, RXD (in serial port mode 0), PWM, CLKOUT, ALE, $\overline{BHE}$, $\overline{RD}$, $\overline{WR}$, and $\overline{RESET}$ and all pins of HSO and P3 and P4 when used as external memory bus (AD0–AD15).
2. I_{OH} = −20 μA for all pins of P1, for P2.6 and P2.7. I_{OH} = −200 μA for TXD, RXD (in serial port mode 0), PWM, CLKOUT, ALE, $\overline{BHE}$, $\overline{WR}$, and all pins of HSO and P3 and P4 when used as external memory bus (AD0–AD15). P3 and P4, when used as ports, have open-drain outputs.

A/D CONVERTER SPECIFICATIONS

A/D Converter operation is verified only on the 8097, 8397, 8095, 8395.

The absolute conversion accuracy is dependent on the accuracy of VREF. The specifications given below assume adherence to the Operating Conditions section of these data sheets. Testing is done at VREF = 5.120V.

Resolution ..... ±0.001 VREF
Accuracy ..... ±0.004 VREF
Differential nonlinearity ..... ±0.002 VREF max
Integral nonlinearity ..... ±0.004 VREF max
Channel-to-channel matching ..... ±1 LSB
Crosstalk (DC to 100 KHz) ..... −60 dB max

A.C. CHARACTERISTICS

(VCC, VPD = 4.5 to 5.5 Volts; T_A = 0°C to 70°C; fosc = 6.0 to 12.0 MHz)

Test Conditions: Load Capacitance on Output Pins = 80 pF
Oscillator Frequency = 12.00 MHz

TIMING REQUIREMENTS (Other system components must meet these specs.)

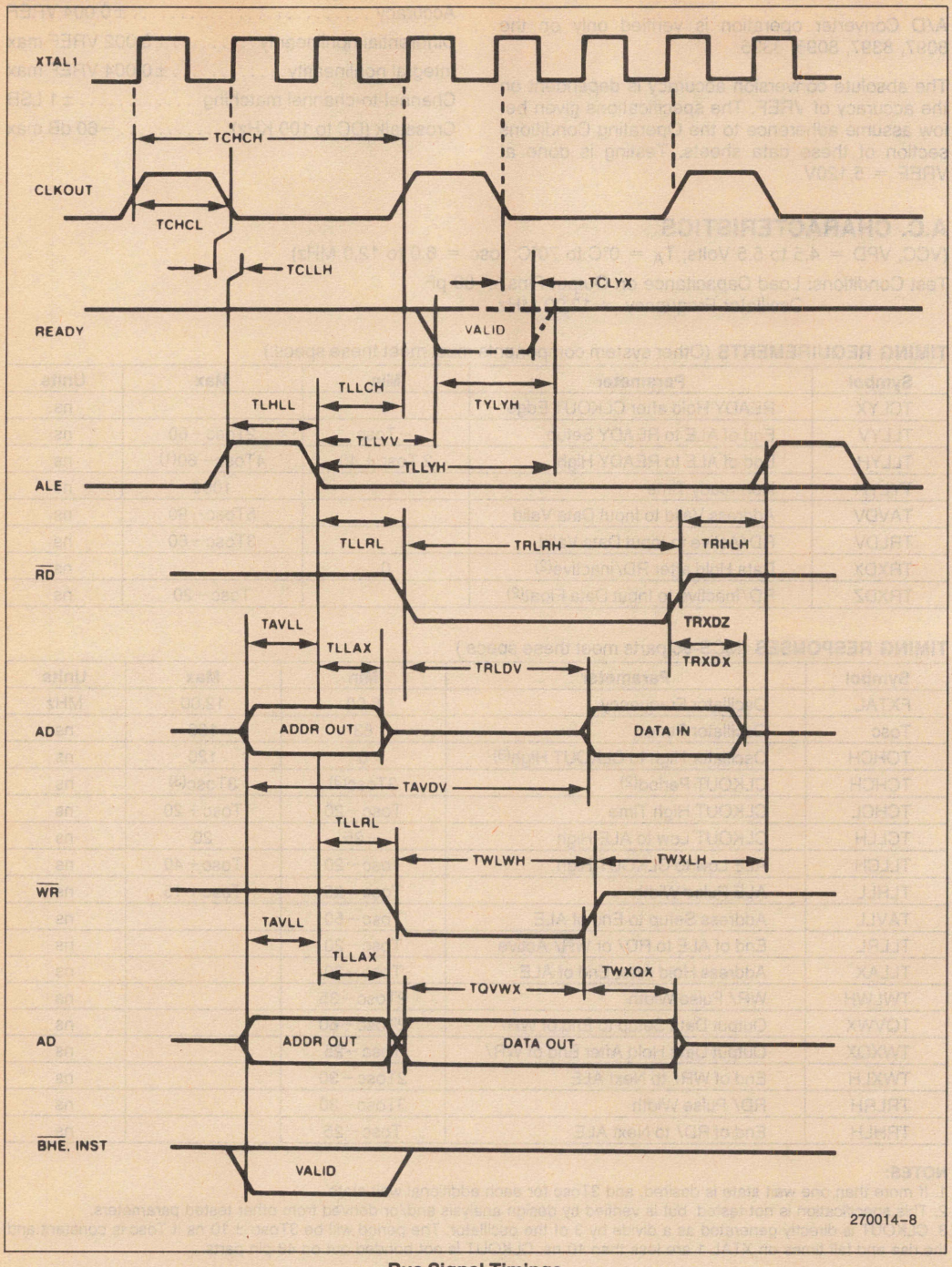
Symbol	Parameter	Min	Max	Units
TCLYX	READY Hold after CLKOUT Edge	0		ns
TLLYV	End of ALE to READY Setup	−Tosc	2Tosc−60	ns
TLLYH	End of ALE to READY High	2 Tosc+40	4Tosc−60(1)	ns
TYLYH	Non-ready Time		1000	ns
TAVDV	Address Valid to Input Data Valid		5Tosc−90	ns
TRLDV	RD/Active to Input Data Valid		3Tosc−60	ns
TRXDX	Data Hold after RD/inactive(2)	0		ns
TRXDZ	RD/Inactive to Input Data Float(2)		Tosc−20	ns

TIMING RESPONSES (MCS-96 parts meet these specs.)

Symbol	Parameter	Min	Max	Units
FXTAL	Oscillator Frequency	6.00	12.00	MHz
Tosc	Oscillator Period	83	166	ns
TOHCH	Oscillator High to CLKOUT High(3)	0	120	ns
TCHCH	CLKOUT Period(2)	3Tosc(3)	3Tosc(3)	ns
TCHCL	CLKOUT High Time	Tosc−20	Tosc+20	ns
TCLLH	CLKOUT Low to ALE High	−25	20	ns
TLLCH	ALE Low to CLKOUT High	Tosc−20	Tosc+40	ns
TLHLL	ALE Pulse Width	Tosc−25	Tosc+15	ns
TAVLL	Address Setup to End of ALE	Tosc−50		ns
TLLRL	End of ALE to RD/ or WR/ Active	Tosc−20		ns
TLLAX	Address Hold After End of ALE	Tosc−20		ns
TWLWH	WR/ Pulse Width	2Tosc−35		ns
TQVWX	Output Data Setup to End of WR/	2Tosc−60		ns
TWXQX	Output Data Hold After End of WR/	Tosc−25		ns
TWXLH	End of WR/ to Next ALE	2Tosc−30		ns
TRLRH	RD/ Pulse Width	3Tosc−30		ns
TRHLH	End of RD/ to Next ALE	Tosc−25		ns

NOTES:
1. If more than one wait state is desired, add 3Tosc for each additional wait state.
2. This specification is not tested, but is verified by design analysis and/or derived from other tested parameters.
3. CLKOUT is directly generated as a divide by 3 of the oscillator. The period will be 3Tosc ±10 ns if Tosc is constant and the rise and fall times on XTAL 1 are less than 10 ns. CLKOUT is not bonded out on 48-pin parts.

WAVEFORM



Bus Signal Timings

MCS®-96
809X-90, 839X-90
Express

- **Extended Temperature Range (−40°C to +85°C)**
- **Burn-In**

The Intel EXPRESS system offers enhancements to the operational specifications of the MCS®-96 family of microcontrollers. These EXPRESS products are designed to meet the needs of those applications whose operating requirements exceed commercial standards.

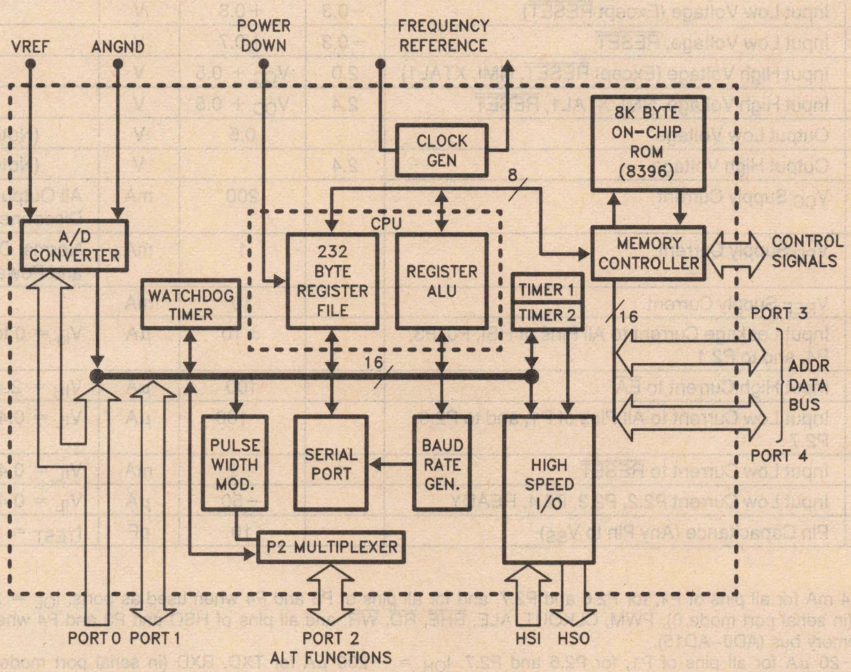
The EXPRESS program includes the commercial standard temperature range with burn-in, and an extended temperature range with or without burn-in.

With the commercial standard temperature range operational characteristics are guaranteed over the temperature range of 0°C to 70°C. With the extended temperature range option, operational characteristics are guaranteed over the range of −40°C to +85°C.

The optional burn-in is dynamic, for a minimum time of 160 hours at 125°C with V_{CC} = 5.5V ±0.5V, following guidelines in MIL-STD-883, Method 1015.

Package types and EXPRESS versions are identified by a one- or two-letter prefix to the part number. The prefixes are listed in Table 1.

This data sheet specifies the parameters for the extended temperature range option. The commercial temperature range data sheets are applicable otherwise.



MCS-96 Block Diagram

November 1985
Order Number: 270104-001

ELECTRICAL CHARACTERISTICS
ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias . −40°C to +85°C
Storage Temperature ..... −40°C to +150°C
Voltage from Any Pin to
VSS or ANGND ..... −0.3V to +7.0V
Average Output Current from Any Pin ..... 10 mA
Power Dissipation 1.5W

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

NOTICE: Specifications contained within the following tables are subject to change.

OPERATING CONDITIONS

Symbol	Parameter	Min	Max	Units
T_A	Ambient Temperature Under Bias	−40	+85	°C
V_{CC}	Digital Supply Voltage	4.5	5.5	V
V_{REF}	Analog Supply Voltage	4.5	5.5	V
f_{OSC}	Oscillator Frequency	6.0	12	MHz
V_{PD}	Power-Down Supply Voltage	4.5	5.5	V

NOTE:
V_{BB} should be connected to ANGND through a 0.01 μF capacitor. ANGND and V_{SS} should be nominally at the same potential.

D.C. CHARACTERISTICS T_A = −40°C to +85°C

Symbol	Parameter	Min	Max	Units	Test Conditions
V_{IL}	Input Low Voltage (Except $\overline{RESET}$)	−0.3	+0.8	V	
V_{IL1}	Input Low Voltage, $\overline{RESET}$	−0.3	+0.7	V	
V_{IH}	Input High Voltage (Except $\overline{RESET}$, NMI, XTAL1)	2.0	V_{CC} + 0.5	V	
V_{IH1}	Input High Voltage, NMI, XTAL1, $\overline{RESET}$	2.4	V_{CC} + 0.5	V	
V_{OL}	Output Low Voltage		0.5	V	(Note 1)
V_{OH}	Output High Voltage	2.4		V	(Note 2)
I_{CC}	V_{CC} Supply Current		200	mA	All Outputs Disconnected
I_{PD}	V_{PD} Supply Current		1	mA	Normal Operation and Power-Down
I_{REF}	V_{REF} Supply Current		10	mA	
I_{LI}	Input Leakage Current to All Pins of HSI, P0, P3, P4, and to P2.1		±10	μA	V_{in} = 0 to V_{CC}
I_{IH}	Input High Current to $\overline{EA}$		100	μA	V_{IH} = 2.4V
I_{IL}	Input Low Current to All Pins of P1, and to P2.6, P2.7		−100	μA	V_{IL} = 0.45V
I_{IL1}	Input Low Current to $\overline{RESET}$		−2	mA	V_{IL} = 0.45V
I_{IL2}	Input Low Current P2.2, P2.3, P2.4, READY		−50	μA	V_{IL} = 0.45V
C_s	Pin Capacitance (Any Pin to V_{SS})		10	pF	f_{TEST} = 1 MHz

NOTES:
1. I_{OL} = 0.4 mA for all pins of P1, for P2.6 and P2.7, and for all pins of P3 and P4 when used as ports. I_{OL} = 2.0 mA for TXD, RSD (in serial port mode 0), PWM, CLKOUT, ALE, $\overline{BHE}$, $\overline{RD}$, $\overline{WR}$, and all pins of HSO and P3 and P4 when used as external memory bus (AD0–AD15).
2. I_{OH} = −20 μA for all pins of P1, for P2.6 and P2.7. I_{OH} = −200 μA for TXD, RXD (in serial port mode 0), PWM, CLKOUT, ALE, $\overline{BHE}$, $\overline{WR}$, and all pins of HSO and P3 and P4 when used as external memory bus (AD0–AD15). P3 and P4, when used as ports, have open-drain outputs.

A/D CONVERTER SPECIFICATIONS

A/D Converter operation is verified only on the 8097, 8397, 8095, 8395.

The absolute conversion accuracy is dependent on the accuracy of V_{REF}. The specifications given below assume adherence to the Operating Conditions section of these data sheets. Testing is done at V_{REF} = 5.120V.

Resolution ..... $\pm 0.001\ V_{REF}$
Accuracy ..... $\pm 0.004\ V_{REF}$
Differential nonlinearity ..... $\pm 0.002\ V_{REF}$ max
Integral nonlinearity ..... $\pm 0.004\ V_{REF}$ max

A.C. CHARACTERISTICS

V_{CC}, V_{PD} = 4.5V to 5.5V, T_A = −40°C to +85°C; f_{osc} = 6.0 MHz to 12.0 MHz
Test Conditions: Load capacitance on output pins = 80 pF
Oscillator Frequency = 12.00 MHz

TIMING REQUIREMENTS Other system components must meet these specs

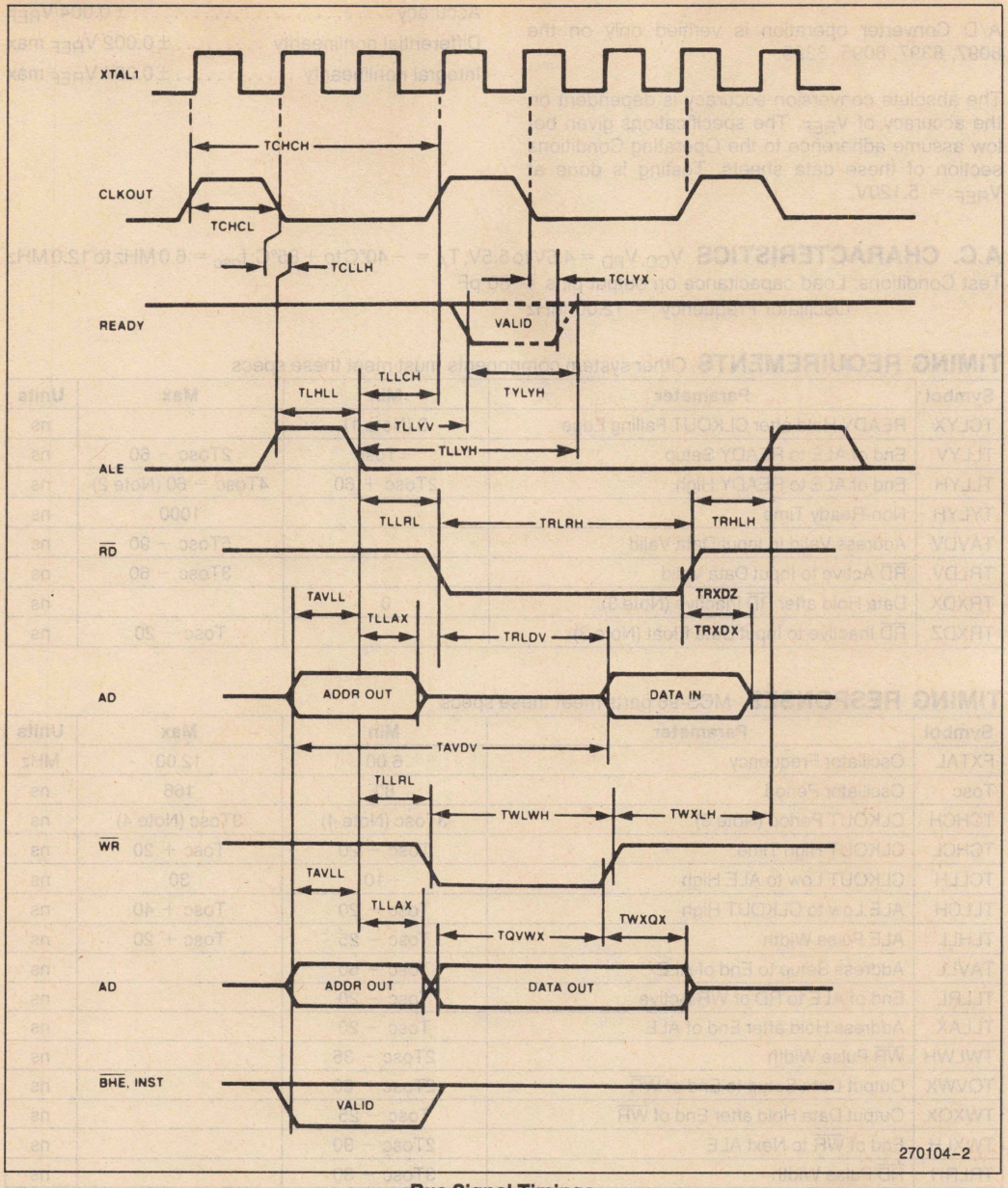
Symbol	Parameter	Min	Max	Units
TCLYX	READY Hold after CLKOUT Falling Edge	0 (Note 1)		ns
TLLYV	End of ALE to READY Setup	−Tosc	2Tosc − 60	ns
TLLYH	End of ALE to READY High	2Tosc + 60	4Tosc − 60 (Note 2)	ns
TYLYH	Non-Ready Time		1000	ns
TAVDV	Address Valid to Input Data Valid		5Tosc − 90	ns
TRLDV	$\overline{RD}$ Active to Input Data Valid		3Tosc − 60	ns
TRXDX	Data Hold after $\overline{RD}$ Inactive (Note 3)	0		ns
TRXDZ	$\overline{RD}$ Inactive to Input Data Float (Note 3)		Tosc − 20	ns

TIMING RESPONSES MCS-96 parts meet these specs

Symbol	Parameter	Min	Max	Units
FXTAL	Oscillator Frequency	6.00	12.00	MHz
Tosc	Oscillator Period	83	166	ns
TCHCH	CLKOUT Period (Note 3)	3Tosc (Note 4)	3Tosc (Note 4)	ns
TCHCL	CLKOUT High Time	Tosc − 20	Tosc + 20	ns
TCLLH	CLKOUT Low to ALE High	−10	30	ns
TLLCH	ALE Low to CLKOUT High	Tosc − 20	Tosc + 40	ns
TLHLL	ALE Pulse Width	Tosc − 25	Tosc + 20	ns
TAVLL	Address Setup to End of ALE	Tosc − 50		ns
TLLRL	End of ALE to $\overline{RD}$ or $\overline{WR}$ Active	Tosc − 20		ns
TLLAX	Address Hold after End of ALE	Tosc − 20		ns
TWLWH	$\overline{WR}$ Pulse Width	2Tosc − 35		ns
TQVWX	Output Data Setup to End of $\overline{WR}$	2Tosc − 60		ns
TWXQX	Output Data Hold after End of $\overline{WR}$	Tosc − 25		ns
TWXLH	End of $\overline{WR}$ to Next ALE	2Tosc − 30		ns
TRLRH	$\overline{RD}$ Pulse Width	3Tosc − 30		ns
TRHLH	End of $\overline{RD}$ to Next ALE	Tosc − 30		ns

NOTES:
1. If the 48-pin part is being used then this timing can be generated by assuming that the CLKOUT falling edge has occurred at 2Tosc + 60 (TLLCH(max) + TCHCL(max)) after the falling edge of ALE.
2. If more than one wait state is desired, add 3Tosc for each additional wait state.
3. This specification is not tested, but is verified by design analysis and/or derived from other tested parameters.
4. CLKOUT is directly generated as a divide by 3 of the oscillator. The period will be 3Tosc ±10 ns if Tosc is constant and the rise and fall times on XTAL 1 are less than 10 ns.

WAVEFORM

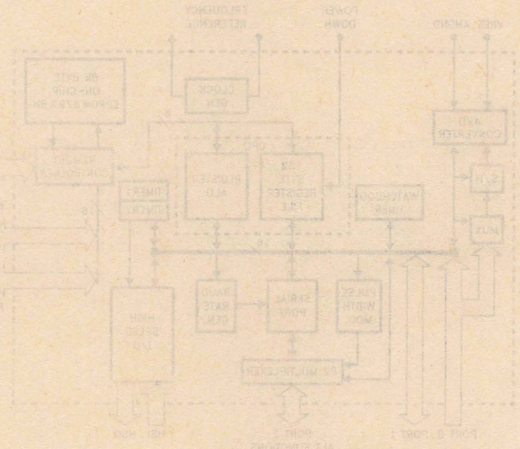


Bus Signal Timings

Table 1. MCS®-96 Prefix Identification

Prefix	Package Type	Temperature Range	Burn-In
A	Ceramic PGA-68L	Commercial	No
N	PLCC-68L	Commercial	No
C	Ceramic DIP-48L	Commercial	No
TA	Ceramic PGA-68L	Extended	No
TN	PLCC-68L	Extended	No
TC	Ceramic DIP-48L	Extended	No
QA	Ceramic PGA-68L	Commercial	Yes
QN	PLCC-68L	Commercial	Yes
QC	Ceramic DIP-48L	Commercial	Yes
LA	Ceramic PGA-68L	Extended	Yes
LN	PLCC-68L	Extended	Yes
LC	Ceramic DIP-48L	Extended	Yes

EXAMPLES:
A8097-90 indicates an 8097-90 in a ceramic pin grid array package specified for commercial temperature without burn-in.
LC8095-90 indicates an 8095-90 in a ceramic DIP package specified for extended temperature range with burn-in.



MCS®-96
809XBH, 839XBH, 879XBH
ADVANCED 16-BIT MICROCONTROLLER WITH 8- OR 16-BIT EXTERNAL BUS

- 879XBH: an 809XBH with 8K Bytes of On-Chip EPROM
- 839XBH: an 809XBH with 8K Bytes of On-Chip ROM

- 232 Byte Register File
- Register-to-Register Architecture
- 10-Bit A/D Converter with S/H
- Five 8-Bit I/O Ports
- 20 Interrupt Sources
- Pulse-Width Modulated Output
- ROM/EPROM Lock
- Run-Time Programmable EPROM
- High Speed I/O Subsystem
- Full Duplex Serial Port
- Dedicated Baud Rate Generator
- 6.25 μs 16 x 16 Multiply
- 6.25 μs 32/16 Divide
- 16-Bit Watchdog Timer
- Four 16-Bit Software Timers
- Two 16-Bit Counter/Timers

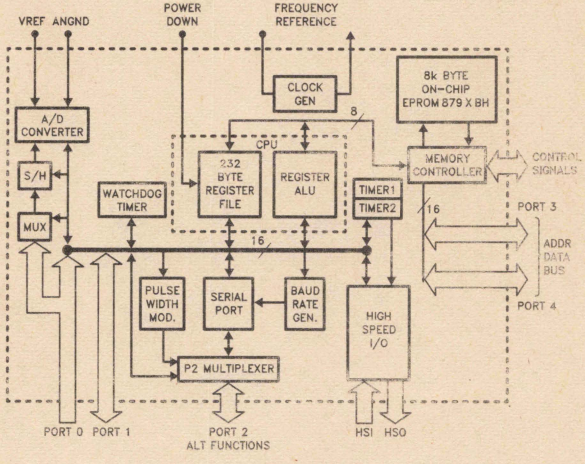
The MCS®-96 family of 16-bit microcontrollers consists of many members, all of which are designed for high-speed control functions. The MCS-96 family members produced using Intel's HMOS-III process are described in this data sheet.

The CPU supports bit, byte, and word operations. Thirty-two bit double-words are supported for a subset of the instruction set. With a 12 MHz input frequency the 8096BH can do a 16-bit addition in 1.0 μs and a 16 x 16-bit multiply or 32/16 divide in 6.25 μs. Instruction execution times average 1 to 2 μs in typical applications.

Four high-speed trigger inputs are provided to record the times at which external events occur. Six high-speed pulse generator outputs are provided to trigger external events at preset times. The high-speed output unit can simultaneously perform software timer functions. Up to four 16-bit software timers can be in operation at once.

The on-chip A/D converter includes a Sample and Hold, and converts up to 8 multiplexed analog input channels to 10-bit digital values. With a 12 MHz crystal, each conversion takes 22 μs. This feature is only available on the 8x95BHs and 8x97BHs, with the 8x95BHs having 4 multiplexed analog inputs.

Also provided on-chip are a serial port, a Watchdog Timer, and a pulse-width modulated output signal.



270090-50

Figure 1. MCS®-96 Block Diagram

December 1986
Order Number: 270090-002

FUNCTIONAL OVERVIEW

The following section is an overview of the 8X9XBH devices, generally referred to as the 8096BH. Additional information is available in the Embedded Controller Handbook, order number 210918.

CPU Architecture

The 8096BH uses the same address space for both program and data memory, except in the address range from 00H through 0FFH. Data fetches in this range are always to the Register File, while instruction fetches from these locations are directed to external memory. (Locations 00H through 0FFH in external memory are reserved for Intel development systems).

Within the Register File, locations 00H through 17H are register mapped I/O control registers, also referred to as Special Function Registers (SFRs). The rest of the Register File (018H through 0FFH) contains 232 bytes of RAM, which can be referenced as bytes, words, or double-words. This register space allows the user to keep the most frequently-used variables in on-chip RAM, which can be accessed faster than external memory. Locations 0F0H through 0FFH can be preserved during power down via a separate power down pin (V_{PD}).

Outside of the Register File, program memory, data memory, and peripherals can be intermixed. The addresses with special significance are:

0000H-	0017H	Register Mapped I/O (SFRs)
0018H-	0019H	Stack Pointer
1FFEH-	1FFFH	Ports 3 and 4
2000H-	2011H	Interrupt Vectors
2012H-	2017H	Reserved
2018H		Chip Configuration Byte
2019H		Reserved
201AH-	201BH	"Jump to Self" Opcode (27 FE)
201CH-	201FH	Reserved
2020H-	202FH	Security Key
2030H-	207FH	Reserved
2080H		Reset Location

The 839XBH carries 8K bytes of ROM, while the 879XBH has 8K bytes of EPROM. With ROM and EPROM parts, the internal program memory occupies addresses 2000H through 3FFFH. Instruction or data fetches from these addresses access the on-chip memory if the $\overline{EA}$ pin is externally held at 5V. If the $\overline{EA}$ pin is at 0V, these addresses access off-chip memory. On the 879XBH parts, holding $\overline{EA}$ at +12.75V puts the part in Programming Mode, which is described in the EPROM Characteristics Section of this data sheet.

A memory map for the MCS-96 product family is shown in Figure 2.

The RALU (Register/ALU) section consists of a 17-bit ALU, the Program Status Word, the Program Counter, and several temporary registers. A key feature of the 8096BH is that it does not use an accumulator. Rather, it operates directly on any register in the Register File. Being able to operate directly on data in the Register File without having to move it into and out of an accumulator results in a significant improvement in execution speed.

In addition to the normal arithmetic and logical functions, the MCS-96 instruction set provides the following special features:

- 6.25 μs Multiply and Divide
- Multiple Shift Instruction
- 3 Operand Instructions
- Normalize Instruction
- Software Reset Instruction

All operations on the 8096BH take place in a set number of "State Times." The 8096BH uses a three phase internal clock, so each state time is 3 oscillator periods. With a 12 MHz clock, each state time requires 0.25 μs.

Operating Modes

The 8096BH supports a variety of options to simplify memory systems, interfacing requirements and ready control. Bus flexibility is provided by allowing selection of bus control signal definitions and run-time selection of the external bus width. In addition, several ready control modes are available to simplify the external hardware requirements for accessing slow devices. The Chip Configuration Register is used to store the operating mode information.

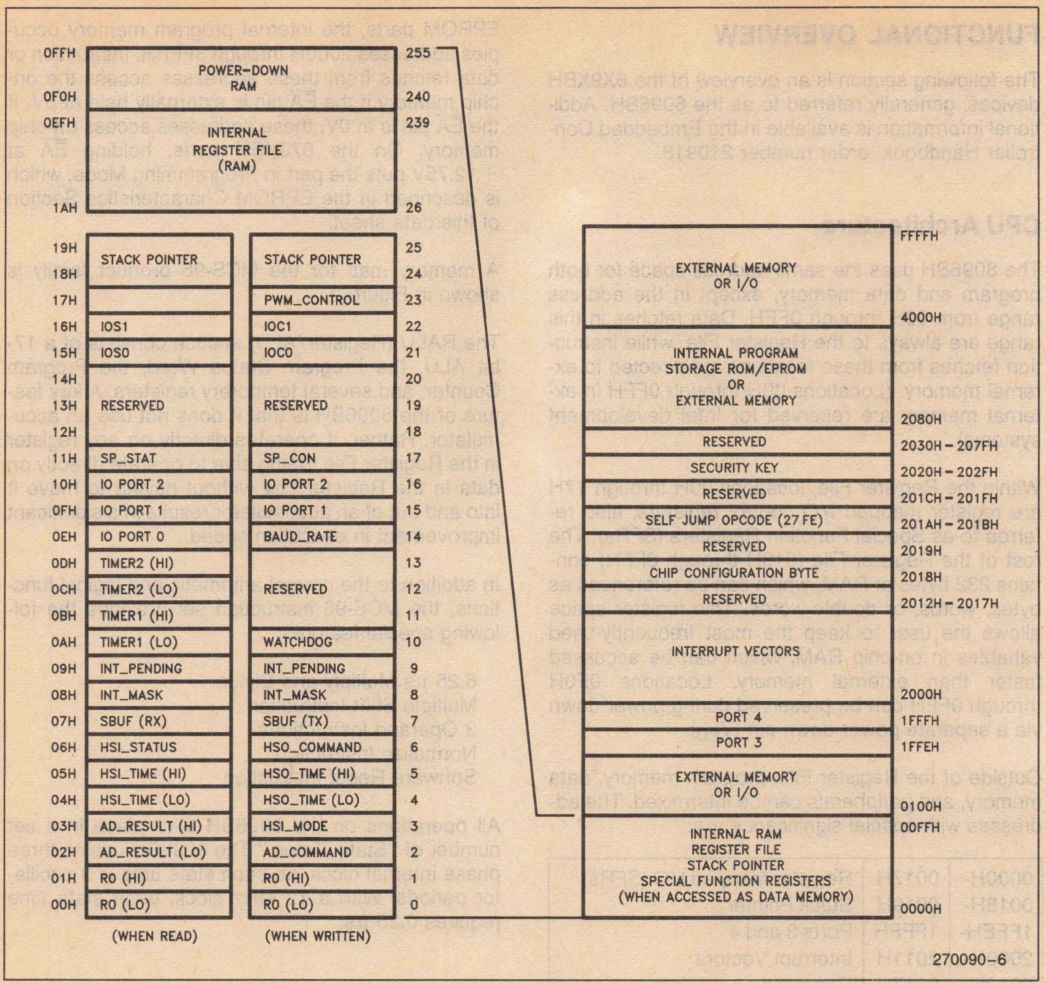


Figure 2. Memory Map

CHIP CONFIGURATION REGISTER (CCR)

Configuration information is stored in the Chip Configuration Register (CCR). Four of the bits in the register specify the bus control mode and ready control mode. Two bits also govern the level of ROM/EPROM protection and one bit is NANDed with the BUSWIDTH pin every bus cycle to determine the bus size. The CCR bit map is shown in Figure 3, and the functions associated with each bit are described later.

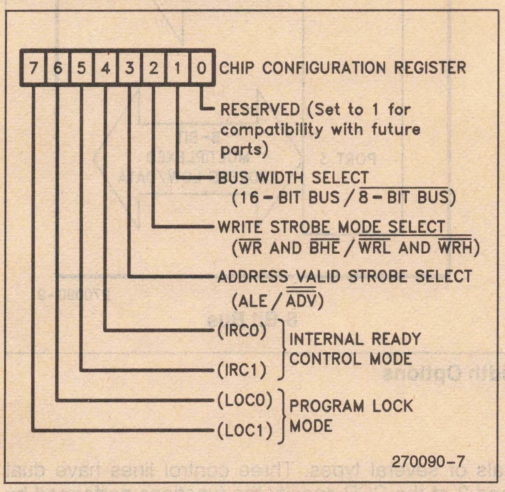


Figure 3. Chip Configuration Register

The CCR is loaded on reset with the Chip Configuration Byte, located at address 2018H. The CCR register is a non-memory mapped location that can only be written to during the reset sequence; once it is loaded it cannot be changed until the next reset occurs. The 8096BH will correctly read this location in every bus mode.

If the $\overline{EA}$ pin is set to a logical 0, the access to 2018H comes from external memory. If $\overline{EA}$ is a logical 1, the access comes from internal ROM/EPROM. If $\overline{EA}$ is +12.5V, the CCR is loaded with a byte from a separate non-memory-mapped location called PCCB (Programming CCB). The Programming Mode is described in the EPROM Characteristics Section.

BUS WIDTH

The 8096BH external bus width can be run-time configured to operate as a standard 16-bit multiplexed address/data bus, or as an 8088 minimum mode type 16-bit address/ 8-bit data bus.

During 16-bit bus cycles, Ports 3 and 4 contain the address multiplexed with data using ALE to latch the address. In 8-bit bus cycles, Port 3 is multiplexed address/data while Port 4 is address bits 8 through 15. The address bits on Port 4 are valid throughout an 8-bit bus cycle. Figure 4 shows the two options.

The bus width can be changed each bus cycle and is controlled using bit 1 of the CCR with the BUSWIDTH pin. If either CCR.1 or BUSWIDTH is a 0, external accesses will be over a 16-bit address/8-bit data bus. If both CCR.1 and BUSWIDTH are 1s, external accesses will be over a 16-bit address/16-bit data bus. Internal accesses are always 16-bits wide.

The bus width can be changed every external bus cycle if a 1 was loaded into CCR bit 1 at reset. If this is the case, changing the value of the BUSWIDTH pin at run-time will dynamically select the bus width. For example, the user could feed the INST line into the BUSWIDTH pin, thus causing instruction accesses to be word wide from EPROMs while data accesses are byte wide to and from RAMs. A second example would be to place an inverted version of address bit 15 on the BUSWIDTH pin. This would make half of external memory word wide, while half is byte wide.

Since BUSWIDTH is sampled after address decoding has had time to occur, even more complex memory maps could be constructed. See the timing specifications for an exact description of BUSWIDTH timings. The bus width will be determined by bit 1 of the CCR alone on 48-pin parts since they do not have a BUSWIDTH pin.

When using an 8-bit bus, some performance degradation is to be expected. On the 8096BH, instruction execution times with an 8-bit bus will slow down if any of three conditions occur. First, word writes to external memory will cause the executing instruction to take two extra state times to complete. Second, word reads from external memory will cause a one state time extension of instruction execution time. Finally, if the prefetch queue is empty when an instruction fetch is requested, instruction execution is lengthened by one state time for each byte that must be externally acquired (worst case is the number of bytes in the instruction minus one).

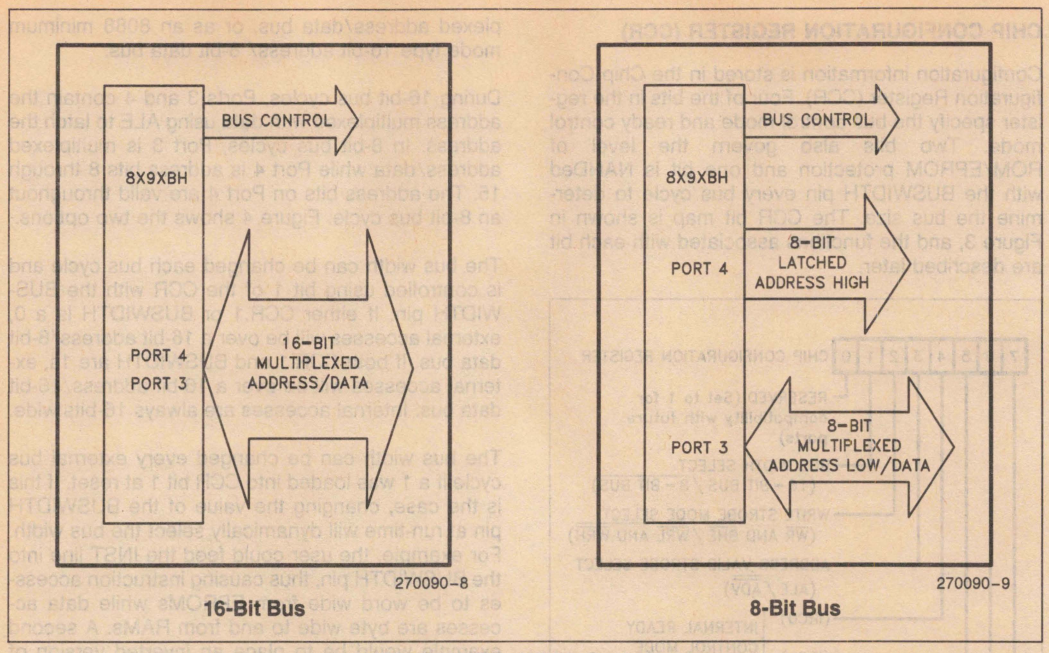


Figure 4. Bus Width Options

BUS CONTROL

The 8096BH can be made to provide bus control signals of several types. Three control lines have dual functions designed to reduce external hardware. Bits 2 and 3 of the CCR specify the functions performed by these control lines.

Standard Bus Control

If CCR bits 2 and 3 are 1s, then the standard 8096BH control signals $\overline{WR}$, $\overline{BHE}$ and ALE are provided (Figure 5). $\overline{WR}$ will come out for every write. $\overline{BHE}$ will be valid throughout the bus cycle and can be combined with $\overline{WR}$ and address line 0 to form $\overline{WRL}$ and $\overline{WRH}$. ALE will rise as the address starts to come out, and will fall to provide the signal to externally latch the address.

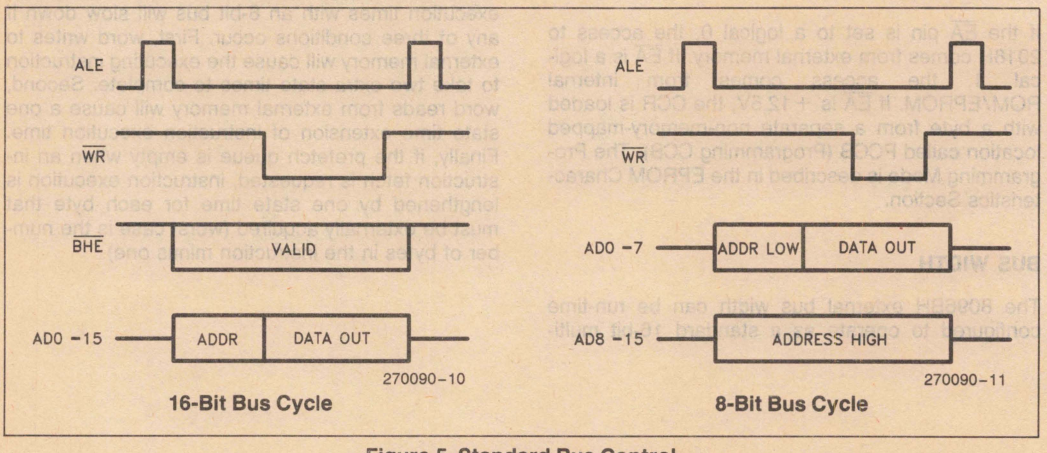


Figure 5. Standard Bus Control

Write Strobe Mode

The Write Strobe Mode eliminates the necessity to externally decode for odd or even byte writes. If CCR bit 2 is a 0, and the bus is in a 16-bit cycle, $\overline{WRL}$ and $\overline{WRH}$ signals are provided in place of $\overline{WR}$ and $\overline{BHE}$ (Figure 6). $\overline{WRL}$ will go low for all byte writes to an even address and all word writes. $\overline{WRH}$ will go low for all byte writes to an odd address and all word writes.

In an 8-bit bus cycle with CCR bit 2 a 0, $\overline{WR}$ is provided (Figure 6). $\overline{WR}$ will go low for all writes.

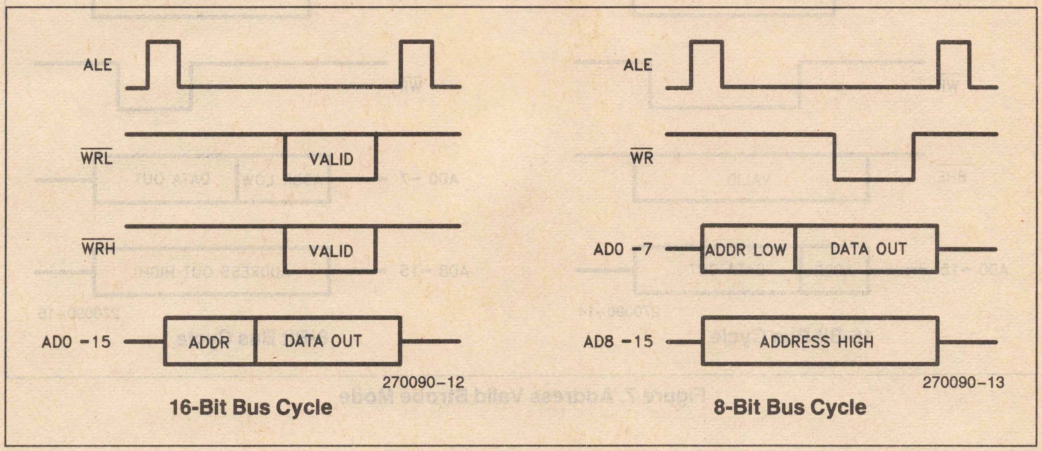


Figure 6. Write Strobe Mode

Address Valid Strobe Mode

If CCR bit 3 is a 0, then an Address Valid Strobe is provided in the place of ALE (Figure 7). When the Address Valid Mode is selected, $\overline{ADV}$ will go low after an external address is set up. It will stay low until the end of the bus cycle, where it will go inactive high. This can be used to provide a chip select for external memory.

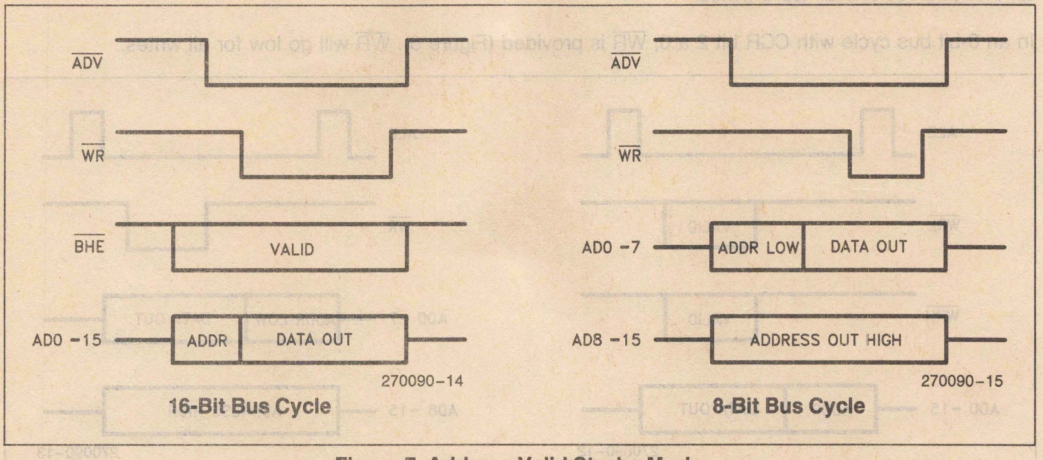


Figure 7. Address Valid Strobe Mode

Address Valid with Write Strobe

If both CCR bits 2 and 3 are 0s, both the Address Valid Strobe and the Write Strobes will be provided for bus control. Figure 8 shows these signals.

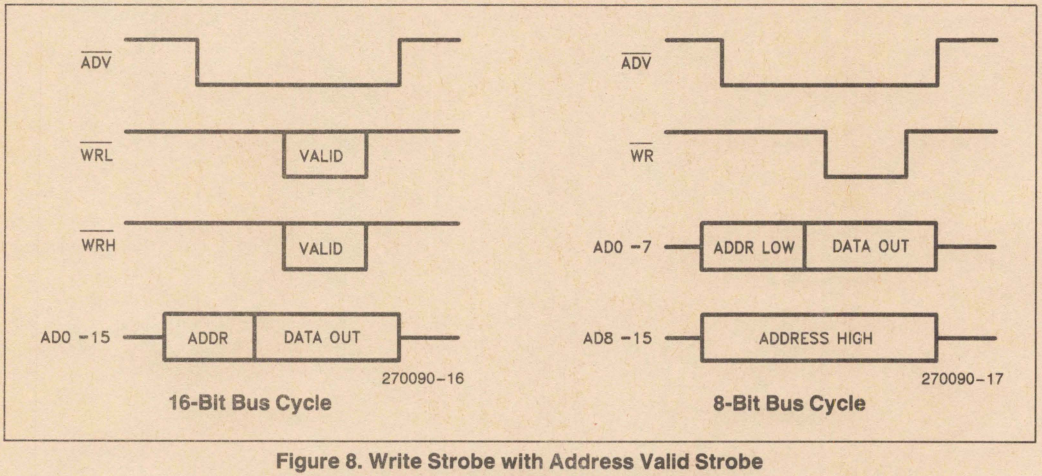


Figure 8. Write Strobe with Address Valid Strobe

READY CONTROL

To simplify ready control, four modes of internal ready control logic have been provided. The modes are chosen by properly configuring bits 4 and 5 of the CCR.

The internal ready control logic can be used to limit the number of wait states that slow devices can insert into the bus cycle. When the READY pin is pulled low, wait states will be inserted into the bus cycle until the READY pin goes high, or the number of wait states equals the number specified by CCR bits 4 and 5, whichever comes first. Table 1 shows the number of wait states that can be selected. Internal ready control can be disabled by loading 11 into bits 4 and 5 of the CCR.

Table 1. Internal Ready Control

IRC1	IRC0	Description
0	0	Limit to 1 Wait State
0	1	Limit to 2 Wait States
1	0	Limit to 3 Wait States
1	1	Disable Internal Ready Control

This feature provides for simple ready control. For example, every slow memory chip select line could be ORed together and be connected to the READY pin with CCR bits 4 and 5 programmed to give the proper number of wait states to the slow devices.

ROM/EPROM LOCK

Four modes of program memory lock are available on the 839XBH and 879XBH parts. CCR bits 6 and 7 (LOC0, LOC1) select whether internal program memory can be read (or written in EPROM parts) by a program executing from external memory. The modes are shown in Table 2. Internal ROM/EPROM addresses 2020H through 3FFFH are protected from reads while 2000H through 3FFFH are protected from writes, as set by the CCR.

Table 2. Program Lock Modes

LOC1	LOC0	Protection
0	0	Read and Write Protected
0	1	Read Protected
1	0	Write Protected
1	1	No Protection

Only code executing from internal memory can read protected internal memory, while a write protected memory can not be written to, even from internal execution. As a result of 8096BH prefetching of instructions, however, accesses to protected memory are not allowed for instructions located above 3FFAH. Note that the interrupt vectors and the CCR are not protected.

To provide ROM/EPROM lock while allowing verification and testing, the 839XBH and 879XBH require security key verification before programming or test modes are allowed to read protected memory. More information on ROM/EPROM Lock can be found in the EPROM Characteristics section.

High Speed I/O Unit (HSIO)

The HSIO unit consists of the High Speed Input Unit (HSI), the High Speed Output Unit (HSO), one counter and one timer. "High Speed" denotes that the units can perform functions related to the timers without CPU intervention. The HSI records times when events occur and the HSO triggers events at pre-programmed times.

All actions within the HSIO unit are synchronized to the timers. The two 16-bit timer/counter registers in the HSIO unit are cleared on chip reset and can be programmed to generate an interrupt on overflow. The Timer 1 register is automatically incremented every 8 state times (every 2.0 μs, with a 12 MHz clock). The Timer 2 register can be programmed to count transitions on either the T2CLK pin or HSI.1 pin. It is incremented on both positive and negative edges of the selected input line. In addition to being cleared by reset, Timer 2 can also be cleared in software or by signals from input pins T2RST or HSI.0. Neither of these timers is required for either the Watchdog Timer or the serial port.

The High Speed Input (HSI) unit can detect transitions on any of its 4 input lines. When one occurs it records the time (from Timer 1) and which input lines made the transition. This information is recorded with 2 μs (12 MHz system) resolution and stored in an 8-level FIFO. The unit can be programmed to look for four types of events, as shown in Figure 9. It can activate the HSI Data Available interrupt either when the Holding Register is loaded or the 6th FIFO entry has been made. Each input line can be individually enabled or disabled to the HSI unit by software.

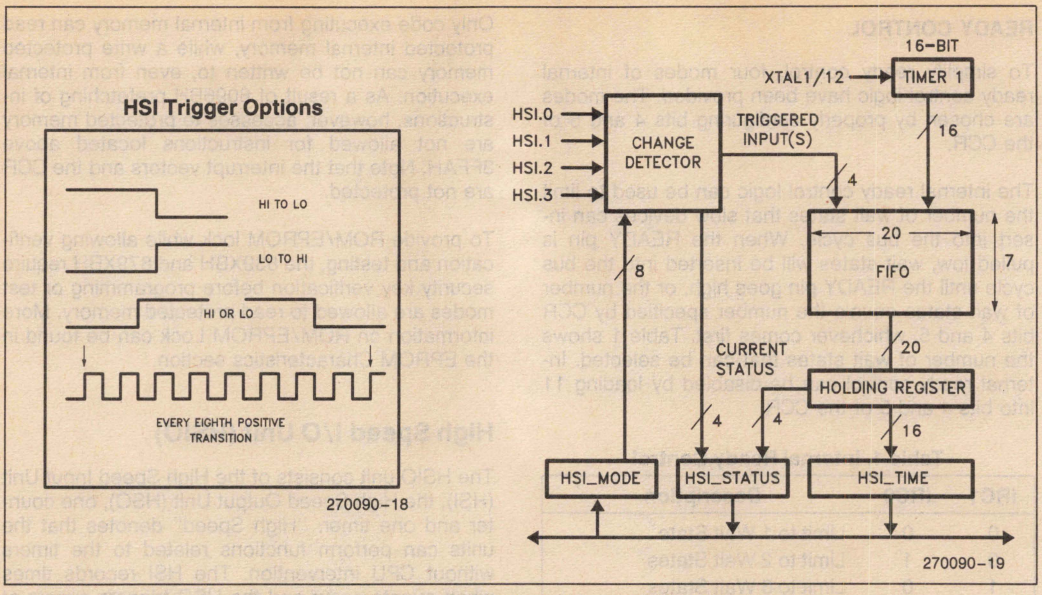


Figure 9. High Speed Input Unit

The High Speed Output (HSO) unit is shown in Figure 10. It can be programmed to set or clear any of its 6 output lines, reset Timer 2, trigger an A/D conversion, or set one of 4 Software Timer flags at a programmed time. An interrupt can be enabled for any of these events. Either Timer 1 or Timer 2 can be referenced for the programmed time value and up to 8 commands for preset actions can be stored in the CAM (Content Addressable Memory) file at any one time. As each action is carried out at its preset time that command is removed from the CAM making space for another command. HSO.4 and HSO.5 are shared with the HSI unit as HSI.2 and HSI.3, and can be individually enabled or disabled as outputs.

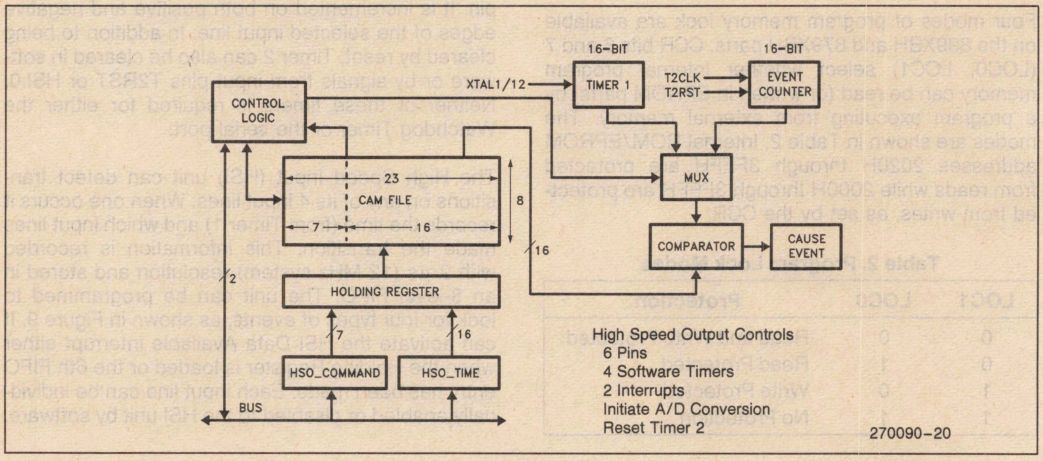


Figure 10. High Speed Output Unit

Standard I/O Ports

There are 5 8-bit I/O ports on the 8096BH in addition to the High Speed I/O lines.

Port 0 is an input-only port which shares its pins with the analog inputs to the A/D converter. The port can be read digitally and/or, by writing to the A/D Command Register, one of the lines can be selected as the input to the A/D converter. Port 0 is also used to input mode information on EPROM parts operating in the Programming Mode.

Port 1 is a quasi-bidirectional I/O port. "Quasi-bidirectional" means the port pin has a weak internal pullup that is always active and an internal pulldown which can either be on (to output a 0) or off (to output a 1). This configuration allows the pin to be used as either an input or an output without using a data direction register. In parallel with the weak internal pullup is a much stronger internal pullup that is activated for one state time when the pin is internally driven from 0 to 1. This is done to speed up the 0-to-1 transition time.

Port 2 is a multi-functional port. Two of the pins (P2.6, 2.7) are quasi-bidirectional while the remaining six are shared with other functions in the 8096BH, as shown in Table 3. Port 2 is also used for control signals by EPROM parts operating in the Programming Mode.

Table 3. Port 2 Pin Functions

Port	Function	Alternate Function
P2.0	Output	TXD (Serial Port Transmit)
P2.1	Input	RXD (Serial Port Receive)
P2.2	Input	EXTINT (External Interrupt)
P2.3	Input	T2CLK (Timer 2 Clock)
P2.4	Input	T2RST (Timer 2 Reset)
P2.5	Output	PWM (Pulse Width Modulation)

Ports 3 and 4 are bi-directional I/O ports with open drain outputs. These pins are also used as the multiplexed address/data bus when accessing external memory, in which case they have strong internal pullups. The internal pullups are only used during external memory read or write cycles when the pins are outputting address or data bits. At any other time, the internal pullups are disabled. When used as a system bus, Ports 3 and 4 can be configured to be either a multiplexed 16-bit address/data bus or a multiplexed 16-bit address/ 8-bit data bus. EPROM parts also use Ports 3 and 4 to pass programming commands, addresses, data and status.

Serial Port

The serial port is compatible with the MCS-51 family, (8051, 8031 etc.), serial port. It is full duplex, and double-buffered on receive. There are 3 asynchronous modes and 1 synchronous mode of operation for the serial port. The asynchronous modes allow for 8 or 9 bits of data with even parity optionally inserted for one of the data bits. Selective interrupts based on the 9th data bit are available to support interprocessor communication.

Baud rates in all modes are determined by an independent 16-bit on-chip baud rate generator. Either the XTAL1 pin or the T2CLK pin can be used as the input to the baud rate generator. The maximum baud rate in the asynchronous mode is 187.5 KBaud. The maximum baud rate in the synchronous mode is 1.5 MBaud.

Pulse Width Modulator (PWM)

The PWM output shares a pin with port bit P2.5. When the PWM output is selected, this pin outputs a pulse train having a fixed period of 256 state times, and a programmable width of 0 to 255 state times. The width is programmed by loading the desired value, in state times, to the PWM Control Register.

A/D Converter with Sample and Hold

The analog-to-digital converter is a 10-bit, successive approximation converter with internal sample and hold. It has a fixed conversion time of 88 state times which includes the 4 state acquisition time of the internal Sample/Hold. With a 12 MHz clock, the conversion takes 22 μs, including the 1 μs sample for the Sample and Hold. The Sample acquisition begins 4 state times after the conversion is triggered. A 2 pF capacitance is charged from the input signal during acquisition.

The analog input must be in the range of 0 to V_{REF} (nominally, V_{REF} = 5V). This input can be selected from 8 analog input lines, which connect to the same pins as Port 0. A conversion can be initiated either by setting a control bit in the A/D Command register, or by programming the HSO unit to trigger the conversion at some specified time.

Interrupts

The 8096BH has 20 interrupt sources which vector through 8 interrupt vectors. A 0-to-1 transition from

any of the sources sets a corresponding bit in the Interrupt Pending register. The content of the Interrupt Mask register determines if a pending interrupt will be serviced or not. If it is to be serviced, the CPU pushes the current Program Counter onto the stack and reloads it with the vector corresponding to the desired interrupt. The interrupt vectors are located in addresses 2000H through 2011H, as shown in Figure 11.

Vector	Vector Location		Priority
	(High Byte)	(Low Byte)	
Software	2011H	2010H	Not Applicable
Extint	200FH	200EH	7 (Highest)
Serial Port	200DH	200CH	6
Software Timers	200BH	200AH	5
HSI.0	2009H	2008H	4
High Speed Outputs	2007H	2006H	3
HSI Data Available	2005H	2004H	2
A/D Conversion Complete	2003H	2002H	1
Timer Overflow	2001H	2000H	0 (Lowest)

Figure 11. Interrupt Vectors

At the end of the interrupt routine the RET instruction pops the program counter from the stack and execution continues where it left off. It is not necessary to store and replace registers during interrupt routines as each routine can be set up to use a different section of the Register File. This feature of the architecture provides for very fast context switching. While the 8096BH has a single priority level in the sense that any interrupt may itself be interrupted, a priority structure exists for resolving simultaneously pending interrupts, as indicated in Figure 11. Since the interrupt pending and interrupt mask registers can be manipulated in software, it is possible to dynamically alter the interrupt priorities to suit the users software.

Watchdog Timer

The Watchdog Timer is a 16-bit counter which, once started, is incremented every state time. If not cleared before it overflows, the $\overline{\text{RESET}}$ pin will be pulled down for two state times, causing the system to be reinitialized. In a 12 MHz system, the Watchdog Timer overflows after 16 ms.

This feature is provided as a means of graceful recovery from a software upset. The counter must be cleared by the software before it overflows, or else the system assumes an upset has occurred and activates $\overline{\text{RESET}}$. Once the Watchdog Timer is started it cannot be turned off by software. The flip-flop which enables the Watchdog Timer has been designed to maintain its state through V_{CC} glitches to as low as 0V or as high as 7V for 1 μs to 1 ms.

To start the Watchdog Timer, or to clear it, one writes 1EH followed by 0E1H to the WDT address (000AH). The Watchdog cannot be stopped once it is started unless the system is reset.

PACKAGING

The 8096BH is available in 48-pin and 68-pin packages, with and without A/D, and with and without on-chip ROM or EPROM. The MCS-96 numbering system is shown in Figure 12. Figures 13–17 show the pinouts for the 48- and 68-pin packages. The 48-pin version is offered in a Dual-In-Line package while the 68-pin versions come in a Plastic Leaded Chip Carrier (PLCC), a Pin Grid Array (PGA) or a Type "B" Leadless Chip Carrier.

		Without A/D	With A/D
ROMless	48 Pin		C8095CH - Ceramic DIP P8095BH - Plastic DIP
	68 Pin	A8096BH - Ceramic PGA N8096BH - PLCC	A8097BH - Ceramic PGA N8097BH - PLCC
ROM	48 Pin		C8395BH - Ceramic DIP P8395BH - Plastic DIP
	68 Pin	A8396BH - Ceramic PGA N8396BH - PLCC	A8397BH - Ceramic PGA N8397BH - PLCC
EPROM	48 Pin		C8795BH - Ceramic DIP
	68 Pin	A8796BH - Ceramic PGA R8796BH - Ceramic LCC	A8797BH - Ceramic PGA R8797BH - Ceramic LCC

Figure 12. The MCS-96 Family Nomenclature

PGA/ LCC	PLCC	Description	PGA/ LCC	PLCC	Description	PGA/ LCC	PLCC	Description
1	9	ACH7/P0.7/PMOD.3	24	54	AD6/P3.6	47	31	P1.6
2	8	ACH6/P0.6/PMOD.2	25	53	AD7/P3.7	48	30	P1.5
3	7	ACH2/P0.2	26	52	AD8/P4.0	49	29	HSO.1
4	6	ACH0/P0.0	27	51	AD9/P4.1	50	28	HSO.0
5	5	ACH1/P0.1	28	50	AD10/P4.2	51	27	HSO.5/HSI.3
6	4	ACH3/P0.3	29	49	AD11/P4.3	52	26	HSO.4/HSI.2
7	3	NMI	30	48	AD12/P4.4	53	25	HSI.1
8	2	$\overline{EA}$	31	47	AD13/P4.5	54	24	HSI.0
9	1	VCC	32	46	AD14/P4.6	55	23	P1.4
10	68	VSS	33	45	AD15/P4.7	56	22	P1.3
11	67	XTAL1	34	44	T2CLK/P2.3	57	21	P1.2
12	66	XTAL2	35	43	READY	58	20	P1.1
13	65	CLKOUT	36	42	T2RST/P2.4	59	19	P1.0
14	64	BUSWIDTH	37	41	$\overline{BHE}/\overline{WRH}$	60	18	TXD/P2.0/PVER/SALE
15	63	INST	38	40	$\overline{WR}/\overline{WRL}$	61	17	RXD/P2.1/PALE
16	62	ALE/$\overline{ADV}$	39	39	PWM/P2.5/$\overline{PDO}$/$\overline{SPROG}$	62	16	$\overline{RESET}$
17	61	$\overline{RD}$	40	38	P2.7	63	15	EXTINT/P2.2/$\overline{PROG}$
18	60	AD0/P3.0	41	37	VPP	64	14	VPD
19	59	AD1/P3.1	42	36	VSS	65	13	VREF
20	58	AD2/P3.2	43	35	HSO.3	66	12	ANGND
21	57	AD3/P3.3	44	34	HSO.2	67	11	ACH4/P0.4/PMOD.0
22	56	AD4/P3.4	45	33	P2.6	68	10	ACH5/P0.5/PMOD.1
23	55	AD5/P3.5	46	32	P1.7			

Figure 13. PGA, PLCC and LCC Function Pinouts

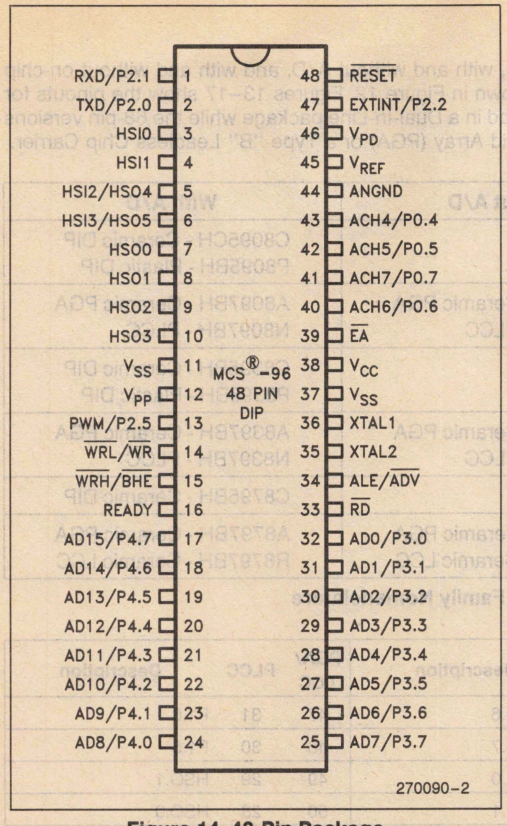


Figure 14. 48-Pin Package

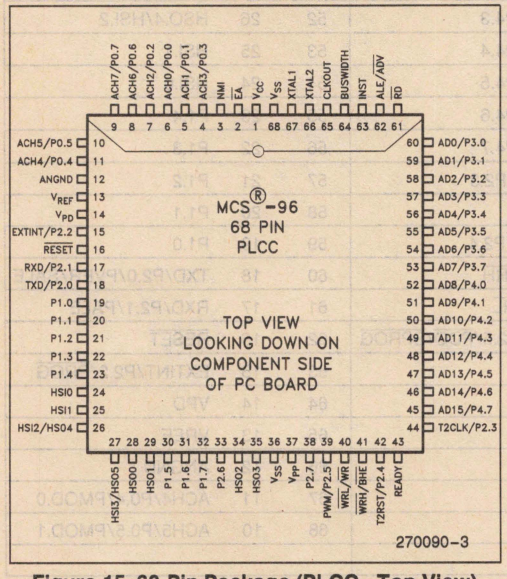


Figure 15. 68-Pin Package (PLCC - Top View)

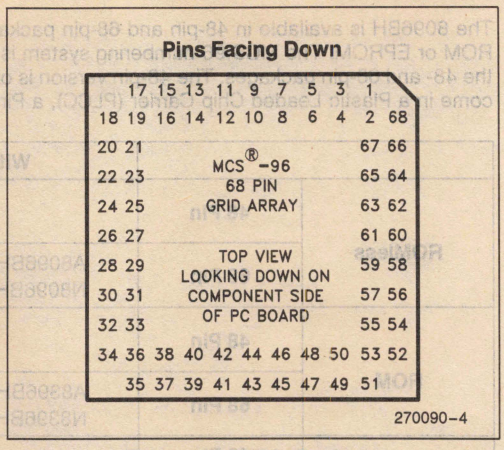


Figure 16. 68-Pin Package (Pin Grid Array - Top View)

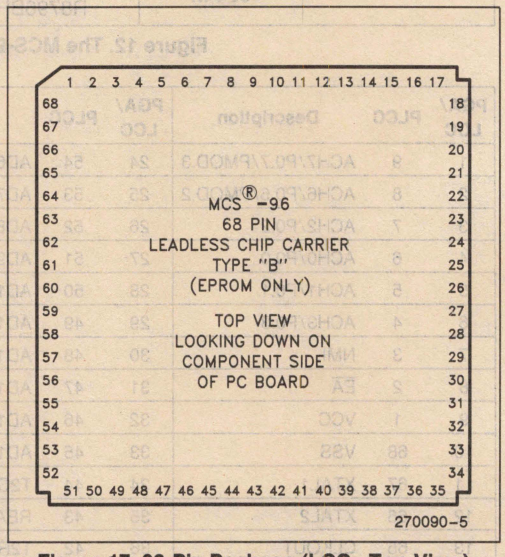


Figure 17. 68-Pin Package (LCC - Top View)

PIN DESCRIPTIONS

Symbol	Name and Function
V_{CC}	Main supply voltage (5V).
V_{SS}	Digital circuit ground (0V). There are two V_{SS} pins, both of which must be connected.
V_{PD}	RAM standby supply voltage (5V). This voltage must be present during normal operation. In a Power Down condition (i.e. V_{CC} drops to zero), if $\overline{RESET}$ is activated before V_{CC} drops below spec and V_{PD} continues to be held within spec., the top 16 bytes in the Register File will retain their contents. $\overline{RESET}$ must be held low during the Power Down and should not be brought high until V_{CC} is within spec and the oscillator has stabilized.
V_{REF}	Reference voltage for the A/D converter (5V). V_{REF} is also the supply voltage to the analog portion of the A/D converter and the logic used to read Port 0. Must be connected for A/D and Port 0 to function.
ANGND	Reference ground for the A/D converter. Must be held at nominally the same potential as V_{SS}.
V_{PP}	Programming voltage for the EPROM parts. It should be +12.75V for programming. This pin is V_{BB} on 8X9X-90 parts. Systems that have this pin connected to ANGND through a capacitance (required on 8X9X-90 parts) do not need to change. Otherwise, tie to V_{CC}.
XTAL1	Input of the oscillator inverter and of the internal clock generator.
XTAL2	Output of the oscillator inverter.
CLKOUT	Output of the internal clock generator. The frequency of CLKOUT is 1/3 the oscillator frequency. It has a 33% duty cycle.
$\overline{RESET}$	Reset input to the chip. Input low for at least 2 state times to reset the chip. The subsequent low-to-high transition re-synchronizes CLKOUT and commences a 10-state-time sequence in which the PSW is cleared, a byte read from 2018H loads CCR, and a jump to location 2080H is executed. Input high for normal operation. $\overline{RESET}$ has an internal pullup.
BUSWIDTH	Input for bus width selection. If CCR bit 1 is a one, this pin selects the bus width for the bus cycle in progress. If BUSWIDTH is a 1, a 16-bit bus cycle occurs. If BUSWIDTH is a 0 an 8-bit cycle occurs. If CCR bit 1 is a 0, the bus is always an 8-bit bus. This pin is the $\overline{TEST}$ pin on 8X9X-90 parts. Systems with $\overline{TEST}$ tied to V_{CC} do not need to change. If this pin is left unconnected, it will rise to V_{CC}.
NMI	A positive transition clears the Watchdog Timer count, and causes a vector to external memory location 0000H. External memory from 00H through 0FFH is reserved for Intel development systems.
INST	Output high during an external memory read indicates the read is an instruction fetch. INST is valid throughout the bus cycle. INST is activated only during external memory accesses.
$\overline{EA}$	Input for memory select (External Access). $\overline{EA}$ equal to a TTL-high causes memory accesses to locations 2000H through 3FFFH to be directed to on-chip ROM/EPROM. $\overline{EA}$ equal to a TTL-low causes accesses to these locations to be directed to off-chip memory. $\overline{EA}$ = +12.5V causes execution to begin in the Programming Mode. $\overline{EA}$ has an internal pulldown, so it goes to 0 unless driven otherwise. $\overline{EA}$ is latched at reset.
ALE/$\overline{ADV}$	Address Latch Enable or Address Valid output, as selected by CCR. Both pin options provide a latch to demultiplex the address from the address/data bus. When the pin is $\overline{ADV}$, it goes inactive high at the end of the bus cycle. $\overline{ADV}$ can be used as a chip select for external memory. ALE/$\overline{ADV}$ is activated only during external memory accesses.
$\overline{RD}$	Read signal output to external memory. $\overline{RD}$ is activated only during external memory reads.
$\overline{WR}$/$\overline{WRL}$	Write and Write Low output to external memory, as selected by the CCR. $\overline{WR}$ will go low for every external write, while $\overline{WRL}$ will go low only for external writes where an even byte is being written. $\overline{WR}$/$\overline{WRL}$ is activated only during external memory writes.
$\overline{BHE}$/$\overline{WRH}$	Bus High Enable or Write High output to external memory, as selected by the CCR. $\overline{BHE}$ = 0 selects the bank of memory that is connected to the high byte of the data bus. A0 = 0 selects the bank of memory that is connected to the low byte of the data bus. Thus accesses to a 16-bit wide memory can be to the low byte only (A0 = 0, $\overline{BHE}$ = 1), to the high byte only (A0 = 1, BHE# = 0), or both bytes (A0 = 0, $\overline{BHE}$ = 0). If the $\overline{WRH}$ function is selected, the pin will go low if the bus cycle is writing to an odd memory location. $\overline{BHE}$/$\overline{WRH}$ is activated only during external memory writes.

PIN DESCRIPTIONS (Continued)

Symbol	Name and Function
READY	Ready input to lengthen external memory cycles, for interfacing to slow or dynamic memory, or for bus sharing. If the pin is high, CPU operation continues in a normal manner. If the pin is low prior to the falling edge of CLKOUT, the memory controller goes into a wait mode until the next positive transition in CLKOUT occurs with READY high. The bus cycle can be lengthened by up to 1 μs. When the external memory is not being used, READY has no effect. Internal control of the number of wait states inserted into a bus cycle held not ready is available through configuration of CCR. READY has a weak internal pullup, so it goes to 1 unless externally pulled low.
HSI	Inputs to High Speed Input Unit. Four HSI pins are available: HSI.0, HSI.1, HSI.2, and HSI.3. Two of them (HSI.2 and HSI.3) are shared with the HSO Unit. The HSI pins are also used as inputs by EPROM parts in Programming Mode.
HSO	Outputs from High Speed Output Unit. Six HSO pins are available: HSO.0, HSO.1, HSO.2, HSO.3, HSO.4, and HSO.5. Two of them (HSO.4 and HSO.5) are shared with the HSI Unit.
Port 0	8-bit high impedance input-only port. These pins can be used as digital inputs and/or as analog inputs to the on-chip A/D converter. These pins are also a mode input to EPROM parts in the Programming Mode.
Port 1	8-bit quasi-bidirectional I/O port.
Port 2	8-bit multi-functional port. Six of its pins are shared with other functions in the 8096BH, the remaining 2 are quasi-bidirectional. These pins are also used to input and output control signals on EPROM parts in Programming Mode.
Ports 3 and 4	8-bit bi-directional I/O ports with open drain outputs. These pins are shared with the multiplexed address/data bus which has strong internal pullups. Ports 3 and 4 are also used as a command, address and data path by EPROM parts operating in the Programming Mode.

INSTRUCTION SET

The 8096BH instruction set makes use of six addressing modes as described below:

DIRECT—The operand is specified by an 8-bit address field in the instruction. The operand must be in the Register File or SFR space (locations 0000H through 00FFH).

IMMEDIATE—The operand itself follows the opcode in the instruction stream as immediate data. The immediate data can be either 8-bits or 16-bits as required by the opcode.

INDIRECT—An 8-bit address field in the instruction gives the word address of a word register in the Register File which contains the 16-bit address of the operand. The operand can be anywhere in memory.

INDIRECT WITH AUTO-INCREMENT—Same as Indirect, except that, after the operand is referenced, the word register that contains the operand's address is incremented by 1 if the operand is a byte, or by 2 if the operand is a word.

INDEXED (LONG AND SHORT)—The instruction contains an 8-bit address field and either an 8-bit or a 16-bit displacement field. The 8-bit address field gives the word address of a word register in the Register File which contains a 16-bit base address. The 8- or 16-bit displacement field contains a signed displacement that will be added to the base address to produce the address of the operand. The operand can be anywhere in memory.

The 8096BH contains a zero register at word address 0000H (and which contains 0000H). This register is available for performing comparisons and for use as a base register in indexed addressing. This effectively provides direct addressing to all 64K of memory.

In the 8096BH, the Stack Pointer is at word address 0018H in the Register File. If the 8-bit address field contains 18H, the Stack Pointer becomes the base register. This allows direct accessing of variables in the stack.

The following tables list the MCS-96 instructions, their opcodes, and execution times.

Instruction Summary

Mnemonic	Operands	Operation (Note 1)	Flags						Notes
			Z	N	C	V	VT	ST	
ADD/ADDB	2	D ← D + A	↕	↕	↕	↕	↑	—	
ADD/ADDB	3	D ← B + A	↕	↕	↕	↕	↑	—	
ADDC/ADDCB	2	D ← D + A + C	↓	↕	↕	↕	↑	—	
SUB/SUBB	2	D ← D − A	↕	↕	↕	↕	↑	—	
SUB/SUBB	3	D ← B − A	↕	↕	↕	↕	↑	—	
SUBC/SUBCB	2	D ← D − A + C − 1	↓	↕	↕	↕	↑	—	
CMP/CMPB	2	D − A	↕	↕	↕	↕	↑	—	
MUL/MULU	2	D, D + 2 ← D * A	—	—	—	—	—	?	2
MUL/MULU	3	D, D + 2 ← B * A	—	—	—	—	—	?	2
MULB/MULUB	2	D, D + 1 ← D * A	—	—	—	—	—	?	3
MULB/MULUB	3	D, D + 1 ← B * A	—	—	—	—	—	?	3
DIVU	2	D ← (D, D + 2)/A, D + 2 ← remainder	—	—	—	↕	↑	—	2
DIVUB	2	D ← (D, D + 1)/A, D + 1 ← remainder	—	—	—	↕	↑	—	3
DIV	2	D ← (D, D + 2)/A, D + 2 ← remainder	—	—	—	?	↑	—	
DIVB	2	D ← (D, D + 1)/A, D + 1 ← remainder	—	—	—	?	↑	—	
AND/ANDB	2	D ← D and A	↕	↕	0	0	—	—	
AND/ANDB	3	D ← B and A	↕	↕	0	0	—	—	
OR/ORB	2	D ← D or A	↕	↕	0	0	—	—	
XOR/XORB	2	D ← D (excl. or) A	↕	↕	0	0	—	—	
LD/LDB	2	D ← A	—	—	—	—	—	—	
ST/STB	2	A ← D	—	—	—	—	—	—	
LDBSE	2	D ← A; D + 1 ← SIGN(A)	—	—	—	—	—	—	3, 4
LDBZE	2	D ← A; D + 1 ← 0	—	—	—	—	—	—	3, 4
PUSH	1	SP ← SP − 2; (SP) ← A	—	—	—	—	—	—	
POP	1	A ← (SP); SP ← SP + 2	—	—	—	—	—	—	
PUSHF	0	SP ← SP − 2; (SP) ← PSW; PSW ← 0000H I ← 0	0	0	0	0	0	0	
POPF	0	PSW ← (SP); SP ← SP + 2; I ← ↕	↕	↕	↕	↕	↕	↕	
SJMP	1	PC ← PC + 11-bit offset	—	—	—	—	—	—	5
LJMP	1	PC ← PC + 16-bit offset	—	—	—	—	—	—	5
BR [indirect]	1	PC ← (A)	—	—	—	—	—	—	
SCALL	1	SP ← SP − 2; (SP) ← PC; PC ← PC + 11-bit offset	—	—	—	—	—	—	5
LCALL	1	SP ← SP − 2; (SP) ← PC; PC ← PC + 16-bit offset	—	—	—	—	—	—	5
RET	0	PC ← (SP); SP ← SP + 2	—	—	—	—	—	—	
J (conditional)	1	PC ← PC + 8-bit offset (if taken)	—	—	—	—	—	—	5
JC	1	Jump if C = 1	—	—	—	—	—	—	5
JNC	1	Jump if C = 0	—	—	—	—	—	—	5
JE	1	Jump if Z = 1	—	—	—	—	—	—	5

NOTES:
1. If the mnemonic ends in "B", a byte operation is performed, otherwise a word operation is done. Operands D, B, and A must conform to the alignment rules for the required operand type. D and B are locations in the Register File; A can be located anywhere in memory.
2. D, D + 2 are consecutive WORDS in memory; D is DOUBLE-WORD aligned.
3. D, D + 1 are consecutive BYTES in memory; D is WORD aligned.
4. Changes a byte to a word.
5. Offset is a 2's complement number.

Instruction Summary (Continued)

Mnemonic	Oper-ands	Operation (Note 1)	Flags						Notes
			Z	N	C	V	VT	ST	
JNE	1	Jump if Z = 0	—	—	—	—	—	—	5
JGE	1	Jump if N = 0	—	—	—	—	—	—	5
JLT	1	Jump if N = 1	—	—	—	—	—	—	5
JGT	1	Jump if N = 0 and Z = 0	—	—	—	—	—	—	5
JLE	1	Jump if N = 1 or Z = 1	—	—	—	—	—	—	5
JH	1	Jump if C = 1 and Z = 0	—	—	—	—	—	—	5
JNH	1	Jump if C = 0 or Z = 1	—	—	—	—	—	—	5
JV	1	Jump if V = 1	—	—	—	—	—	—	5
JNV	1	Jump if V = 0	—	—	—	—	—	—	5
JVT	1	Jump if VT = 1; Clear VT	—	—	—	—	0	—	5
JNVT	1	Jump if VT = 0; Clear VT	—	—	—	—	0	—	5
JST	1	Jump if ST = 1	—	—	—	—	—	—	5
JNST	1	Jump if ST = 0	—	—	—	—	—	—	5
JBS	3	Jump if Specified Bit = 1	—	—	—	—	—	—	5, 6
JBC	3	Jump if Specified Bit = 0	—	—	—	—	—	—	5, 6
DJNZ	1	D ← D − 1; if D ≠ 0 then PC ← PC + 8-bit offset	—	—	—	—	—	—	5
DEC/DECB	1	D ← D − 1	↙	↙	↙	↙	↑	—	
NEG/NEGB	1	D ← 0 − D	↙	↙	↙	↙	↑	—	
INC/INCB	1	D ← D + 1	↙	↙	↙	↙	↑	—	
EXT	1	D ← D; D + 2 ← Sign (D)	↙	↙	0	0	—	—	2
EXTB	1	D ← D; D + 1 ← Sign(D)	↙	↙	0	0	—	—	3
NOT/NOTB	1	D ← Logical Not (D)	↙	↙	0	0	—	—	
CLR/CLRB	1	D ← 0	1	0	0	0	—	—	
SHL/SHLB/SHLL	2	C ← msb — — — — — lsb ← 0	↙	?	↙	↙	↑	—	7
SHR/SHRB/SHRL	2	0 → msb — — — — — lsb → C	↙	?	↙	0	—	↙	7
SHRA/SHRAB/SHRAL	2	msb → msb — — — — — lsb → C	↙	↙	↙	0	—	↙	7
SETC	0	C ← 1	—	—	1	—	—	—	
CLRC	0	C ← 0	—	—	0	—	—	—	
CLRVT	0	VT ← 0	—	—	—	—	0	—	
RST	0	PC ← 2080H	0	0	0	0	0	0	8
DI	0	Disable All Interrupts (I ← 0)	—	—	—	—	—	—	
EI	0	Enable All Interrupts (I ← 1)	—	—	—	—	—	—	
NOP	0	PC ← PC + 1	—	—	—	—	—	—	
SKIP	0	PC ← PC + 2	—	—	—	—	—	—	
NORML	2	Left shift till msb = 1; D ← shift count	↙	?	0	—	—	—	7
TRAP	0	SP ← SP − 2; (SP) ← PC PC ← (2010H)	—	—	—	—	—	—	9

NOTES:

1. If the mnemonic ends in "B", a byte operation is performed, otherwise a word operation is done. Operands D, B and A must conform to the alignment rules for the required operand type. D and B are locations in the Register File; A can be located anywhere in memory.
5. Offset is a 2's complement number.
6. Specified bit is one of the 2048 bits in the register file.
7. The "L" (Long) suffix indicates double-word operation.
8. Initiates a Reset by pulling $\overline{\text{RESET}}$ low. Software should re-initialize all the necessary registers with code starting at 2080H.
9. The assembler will not accept this mnemonic.

Opcode and State Time Listing

MNEMONIC	OPERANDS	DIRECT			IMMEDIATE			INDIRECT* NORMAL			INDIRECT* AUTO-INC.		INDEXED* SHORT			INDEXED* LONG	
		OPCODE	BYTES	STATE TIMES	OPCODE	BYTES	STATE TIMES	OPCODE	BYTES	STATE① TIMES	BYTES	STATE① TIMES	OPCODE	BYTES	STATE① TIMES⑧	BYTES	STATE① TIMES⑧
ARITHMETIC INSTRUCTIONS																	
ADD	2	64	3	4	65	4	5	66	3	6/11	3	7/12	67	4	6/11	5	7/12
ADD	3	44	4	5	45	5	6	46	4	7/12	4	8/13	47	5	7/12	6	8/13
ADDB	2	74	3	4	75	3	4	76	3	6/11	3	7/12	77	4	6/11	5	7/12
ADDB	3	54	4	5	55	4	5	56	4	7/12	4	8/13	57	5	7/12	6	8/13
ADDC	2	A4	3	4	A5	4	5	A6	3	6/11	3	7/12	A7	4	6/11	5	7/12
ADDCB	2	B4	3	4	B5	3	4	B6	3	6/11	3	7/12	B7	4	6/11	5	7/12
SUB	2	68	3	4	69	4	5	6A	3	6/11	3	7/12	6B	4	6/11	5	7/12
SUB	3	48	4	5	49	5	6	4A	4	7/12	4	8/13	4B	5	7/12	6	8/13
SUBB	2	78	3	4	79	3	4	7A	3	6/11	3	7/12	7B	4	6/11	5	7/12
SUBB	3	58	4	5	59	4	5	5A	4	7/12	4	8/13	5B	5	7/12	6	8/13
SUBC	2	A8	3	4	A9	4	5	AA	3	6/11	3	7/12	AB	4	6/11	5	7/12
SUBCB	2	B8	3	4	B9	3	4	BA	3	6/11	3	7/12	BB	4	6/11	5	7/12
CMP	2	88	3	4	89	4	5	8A	3	6/11	3	7/12	8B	4	6/11	5	7/12
CMPB	2	98	3	4	99	3	4	9A	3	6/11	3	7/12	9B	4	6/11	5	7/12
MULU	2	6C	3	25	6D	4	26	6E	3	27/32	3	28/33	6F	4	27/32	5	28/33
MULU	3	4C	4	26	4D	5	27	4E	4	28/33	4	29/34	4F	5	28/33	6	29/34
MULUB	2	7C	3	17	7D	3	17	7E	3	19/24	3	20/25	7F	4	19/24	5	20/25
MULUB	3	5C	4	18	5D	4	18	5E	4	20/25	4	21/26	5F	5	20/25	6	21/26
MUL	2	②	4	29	②	5	30	②	4	31/36	4	32/37	②	5	31/36	6	32/37
MUL	3	②	5	30	②	6	31	②	5	32/37	5	33/38	②	6	32/37	7	33/38
MULB	2	②	4	21	②	4	21	②	4	23/28	4	24/29	②	5	23/28	6	24/29
MULB	3	②	5	22	②	5	22	②	5	24/29	5	25/30	②	6	24/29	7	25/30
DIVU	2	8C	3	25	8D	4	26	8E	3	28/32	3	29/33	8F	4	28/32	5	29/33
DIVUB	2	9C	3	17	9D	3	17	9E	3	20/24	3	21/25	9F	4	20/24	5	21/25
DIV	2	②	4	29	②	5	30	②	4	32/36	4	33/37	②	5	32/36	6	33/37
DIVB	2	②	4	21	②	4	21	②	4	24/28	4	25/29	②	5	24/28	6	25/29

270090-45

NOTES:

*Long indexed and Indirect + instructions have identical opcodes with Short indexed and Indirect modes, respectively. The second byte of instructions using any Indirect or indexed addressing mode specifies the exact mode used. If the second byte is even, use Indirect or Short indexed. If it is odd, use Indirect + or Long indexed. In all cases the second byte of the instruction always specifies an even (word) location for the address referenced.

① Number of state times shown for internal/external operands.

② The opcodes for signed multiply and divide are the opcodes for the unsigned functions with an "FE" appended as a prefix.

⑧ State times shown for 16-bit bus.

Opcode and State Time Listing (Continued)

MNEMONIC	OPERANDS	DIRECT			IMMEDIATE			INDIRECT⑥					INDEXED⑥				
								NORMAL			AUTO-INC.		SHORT			LONG	
		OPCODE	BYTES	STATE TIMES	OPCODE	BYTES	STATE TIMES	OPCODE	BYTES	STATE① TIMES	BYTES	STATE① TIMES	OPCODE	BYTES	STATE① TIMES⑧	BYTES	STATE① TIMES⑧
LOGICAL INSTRUCTIONS																	
AND	2	60	3	4	61	4	5	62	3	6/11	3	7/12	63	4	6/11	5	7/12
AND	3	40	4	5	41	5	6	42	4	7/12	4	8/13	43	5	7/12	6	8/13
ANDB	2	70	3	4	71	3	4	72	3	6/11	3	7/12	73	4	6/11	5	7/12
ANDB	3	50	4	5	51	4	5	52	4	7/12	4	8/13	53	5	7/12	6	8/13
OR	2	80	3	4	81	4	5	82	3	6/11	3	7/12	83	4	6/11	5	7/12
ORB	2	90	3	4	91	3	4	92	3	6/11	3	7/12	93	4	6/11	5	7/12
XOR	2	84	3	4	85	4	5	86	3	6/11	3	7/12	87	4	6/11	5	7/12
XORB	2	94	3	4	95	3	4	96	3	6/11	3	7/12	97	4	6/11	5	7/12
DATA TRANSFER INSTRUCTIONS																	
LD	2	A0	3	4	A1	4	5	A2	3	6/11	3	7/12	A3	4	6/11	5	7/12
LDB	2	B0	3	4	B1	3	4	B2	3	6/11	3	7/12	B3	4	6/11	5	7/12
ST	2	C0	3	4	—	—	—	C2	3	7/11	3	8/12	C3	4	7/11	5	8/12
STB	2	C4	3	4	—	—	—	C6	3	7/11	3	8/12	C7	4	7/11	5	8/12
LDBSE	2	BC	3	4	BD	3	4	BE	3	6/11	3	7/12	BF	4	6/11	5	7/12
LDBZE	2	AC	3	4	AD	3	4	AE	3	6/11	3	7/12	AF	4	6/11	5	7/12
STACK OPERATIONS (internal stack)																	
PUSH	1	C8	2	8	C9	3	8	CA	2	11/15	2	12/16	CB	3	11/15	4	12/16
POP	1	CC	2	12	—	—	—	CE	2	14/18	2	14/18	CF	3	14/18	4	14/18
PUSHF	0	F2	1	8													
POPF	0	F3	1	9													
STACK OPERATIONS (external stack)																	
PUSH	1	C8	2	12	C9	3	12	CA	2	15/19	2	16/20	CB	3	15/19	4	16/20
POP	1	CC	2	14	—	—	—	CE	2	16/20	2	16/20	CF	3	16/20	4	16/20
PUSHF	0	F2	1	12													
POPF	0	F3	1	13													

JUMPS AND CALLS							
MNEMONIC	OPCODE	BYTES	STATES	MNEMONIC	OPCODE	BYTES	STATES
LJMP	E7	3	8	LCALL	EF	3	13/16⑤
SJMP	20-27④	2	8	SCALL	28-2F④	2	13/16⑤
BR[ ]	E3	2	8	RET	F0	1	12/16⑤
				TRAP③	F7	1	21/24

270090-46

NOTES:
① Number of state times shown for internal/external operands.
③ The assembler does not accept this mnemonic.
④ The least significant 3 bits of the opcode are concatenated with the following 8 bits to form an 11-bit, 2's complement, offset for the relative call or jump.
⑤ State times for stack located internal/external.
⑧ State times shown for 16-bit bus.

CONDITIONAL JUMPS

All conditional jumps are 2 byte instructions. They require 8 state times if the jump is taken, 4 if it is not.(8)

MNEMONIC	OPCODE	MNEMONIC	OPCODE	MNEMONIC	OPCODE	MNEMONIC	OPCODE
JC	DB	JE	DF	JGE	D6	JGT	D2
JNC	D3	JNE	D7	JLT	DE	JLE	DA
JH	D9	JV	DD	JVT	DC	JST	D8
JNH	D1	JNV	D5	JNVT	D4	JNST	D0

JUMP ON BIT CLEAR OR BIT SET

These instructions are 3-byte instructions. They require 9 state times if the jump is taken, 5 if it is not.(8)

	BIT NUMBER							
MNEMONIC	0	1	2	3	4	5	6	7
JBC	30	31	32	33	34	35	36	37
JBS	38	39	3A	3B	3C	3D	3E	3F

LOOP CONTROL

MNEMONIC	OPCODE	BYTES	STATE TIMES
DJNZ	E0	3	5/9 STATE TIME (NOT TAKEN/TAKEN)(8)

SINGLE REGISTER INSTRUCTIONS

MNEMONIC	OPCODE	BYTES	STATES(8)	MNEMONIC	OPCODE	BYTES	STATES(8)
DEC	05	2	4	EXT	06	2	4
DECB	15	2	4	EXTB	16	2	4
NEG	03	2	4	NOT	02	2	4
NEGB	13	2	4	NOTB	12	2	4
INC	07	2	4	CLR	01	2	4
INCB	17	2	4	CLRB	11	2	4

SHIFT INSTRUCTIONS

INSTR MNEMONIC	WORD		INSTR MNEMONIC	BYTE		INSTR MNEMONIC	DBL WD		STATE TIMES(8)
	OP	B		OP	B		OP	B	
SHL	09	3	SHLB	19	3	SHLL	0D	3	7 + 1 PER SHIFT(7)
SHR	08	3	SHRB	18	3	SHRL	0C	3	7 + 1 PER SHIFT(7)
SHRA	0A	3	SHRAB	1A	3	SHRAL	0E	3	7 + 1 PER SHIFT(7)

SPECIAL CONTROL INSTRUCTIONS

MNEMONIC	OPCODE	BYTES	STATES(8)	MNEMONIC	OPCODE	BYTES	STATES(8)
SETC	F9	1	4	DI	FA	1	4
CLRC	F8	1	4	EI	FB	1	4
CLRVT	FC	1	4	NOP	FD	1	4
RST(6)	FF	1	166	SKIP	00	2	4

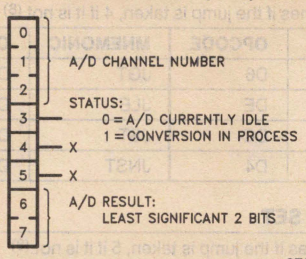
NORMALIZE

MNEMONIC	OPCODE	BYTES	STATE TIMES
NORML	0F	3	11 + 1 PER SHIFT

NOTES:

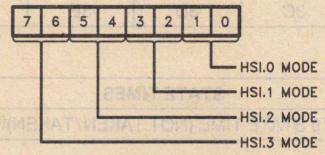
6. This instruction takes 2 states to pull $\overline{\text{RESET}}$ low, then holds it low for 2 states to initiate a reset. The reset takes 12 states, at which time the program restarts at location 2080H. If a capacitor is tied to $\overline{\text{RESET}}$, the pin may take longer to go low and may never reach the V_{OL} specification.
7. Execution will take at least 8 states, even for 0 shift.
8. State times shown for 16-bit bus.

A/D Result LO (02H)



270090-21

HSI_Mode (03H)

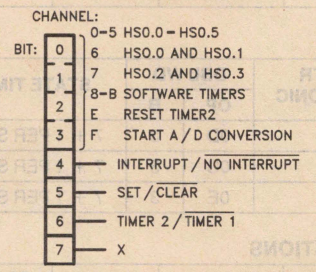


WHERE EACH 2-BIT MODE CONTROL FIELD DEFINES ONE OF 4 POSSIBLE MODES:

00 8 POSITIVE TRANSITIONS
01 EACH POSITIVE TRANSITION
10 EACH NEGATIVE TRANSITION
11 EVERY TRANSITION (POSITIVE AND NEGATIVE)

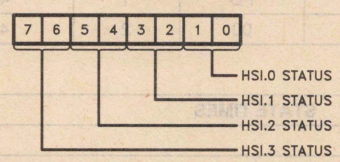
270090-22

HSO Command (06H)



270090-23

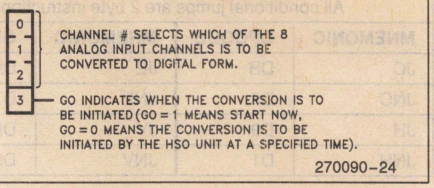
HSI_Status (06H)



WHERE FOR EACH 2-BIT STATUS FIELD THE LOWER BIT INDICATES WHETHER OR NOT AN EVENT HAS OCCURED ON THIS PIN AND THE UPPER BIT INDICATES THE CURRENT STATUS OF THE PIN.

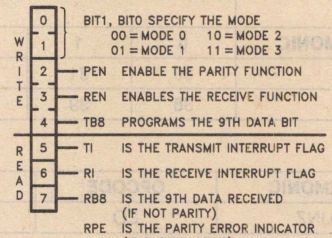
270090-25

A/D Command (02H)



270090-24

SPCON/SPSTAT (11H)



270090-26

Baud Rate Calculations

Using XTAL1:

Mode 0: $\text{Baud Rate} = \frac{\text{XTAL1 frequency}}{4*(B+1)}; B \neq 0$

Others: $\text{Baud Rate} = \frac{\text{XTAL1 frequency}}{64*(B+1)}$

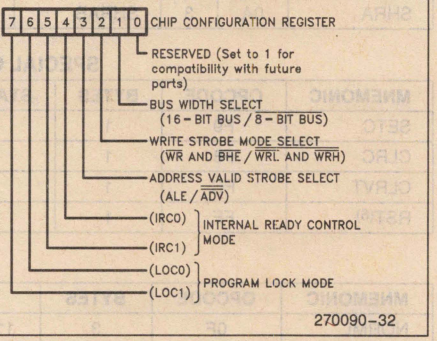
Using T2CLK:

Mode 0: $\text{Baud Rate} = \frac{\text{T2CLK frequency}}{B}; B \neq 0$

Others: $\text{Baud Rate} = \frac{\text{T2CLK frequency}}{16*B}; B \neq 0$

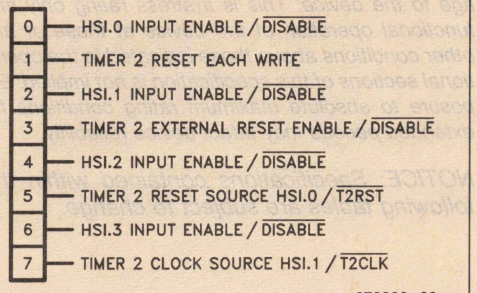
Note that B cannot equal 0, except when using XTAL1 in other than Mode 0.

Chip Configuration

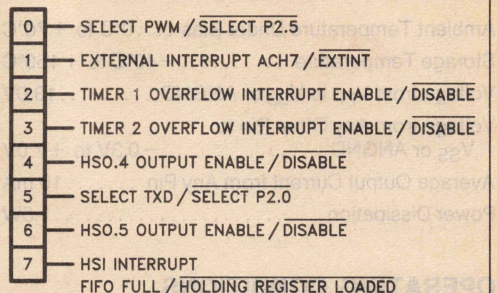


270090-32

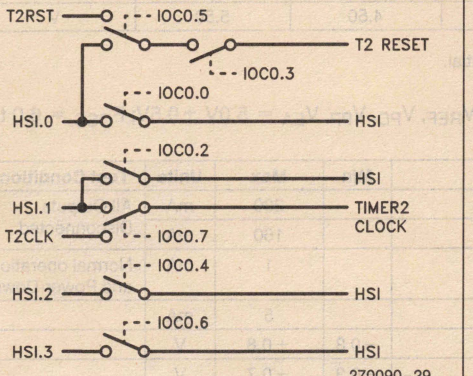
IOC0 (15H)



IOC1 (16H)

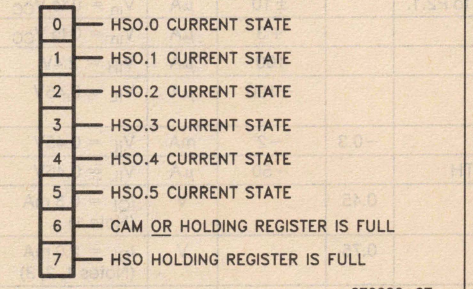


IOC0 (15H)

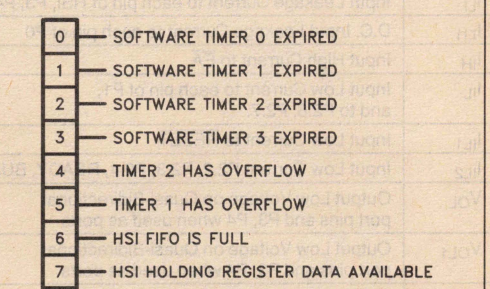


Vector	Vector Location (High Byte)	Vector Location (Low Byte)	Priority
Software	2011H	2010H	Not Applicable
Extint	200FH	200EH	7 (Highest)
Serial Port	200DH	200CH	6
Software Timers	200BH	200AH	5
HSI.0	2009H	2008H	4
High Speed Outputs	2007H	2006H	3
HSI Data Available	2005H	2004H	2
A/D Conversion Complete	2003H	2002H	1
Timer Overflow	2001H	2000H	0 (Lowest)

IOS0 (15H)



IOS1 (16H)



ELECTRICAL CHARACTERISTICS
ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias ....0°C to +70°C
Storage Temperature .....−40°C to +150°C
Voltage from V_{PP} to V_{SS} or ANGND 13.0V
Voltage from Any Other Pin to V_{SS} or ANGND.....−0.3V to +7.0V
Average Output Current from Any Pin .....10 mA
Power Dissipation.....1.5W

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

NOTICE: Specifications contained within the following tables are subject to change.

OPERATING CONDITIONS

Symbol	Parameter	Min	Max	Units
T_A	Ambient Temperature Under Bias	0	+70	C
V_{CC}	Digital Supply Voltage	4.50	5.50	V
V_{REF}	Analog Supply Voltage	4.50	5.50	V
f_{OSC}	Oscillator Frequency	6.0	12	MHz
V_{PD}	Power-Down Supply Voltage	4.50	5.50	V

NOTE: ANGND and V_{SS} should be nominally at the same potential.

D.C. CHARACTERISTICS (Test Conditions: V_{CC}, V_{REF}, V_{PD}, V_{PP}, V_{EA} = 5.0V ±0.5V; F_{OSC} = 6.0 to 12.0 MHz; T_A = 0°C to 70°C; V_{SS}, ANGND = 0V)

Symbol	Parameter	Min	Max	Units	Test Conditions
I_{CC}	V_{CC} Supply Current (0°C ≤ T_A ≤ 70°C		200	mA	All Outputs Disconnected.
I_{CC1}	V_{CC} Supply Current (T_A = 70°C)		160	mA	
I_{PD}	V_{PD} Supply Current		1	mA	Normal operation and Power-Down.
I_{REF}	V_{REF} Supply Current		5	mA	
V_{IL}	Input Low Voltage (Except $\overline{RESET}$)	−0.3	+0.8	V	
V_{IL1}	Input Low Voltage, $\overline{RESET}$	−0.3	+0.7	V	
V_{IH}	Input High Voltage (Except $\overline{RESET}$, NMI, XTAL1)	2.0	V_{CC} +0.5	V	
V_{IH1}	Input High Voltage, $\overline{RESET}$ Rising	2.4	V_{CC} +0.5	V	
V_{IH2}	Input High Voltage, $\overline{RESET}$ Falling	2.1	V_{CC} +0.5	V	
V_{IH3}	Input High Voltage, NMI, XTAL1	2.2	V_{CC} +0.5	V	
I_{LI}	Input Leakage Current to each pin of HSI, P3, P4, and to P2.1.		±10	µA	V_{in} = 0 to V_{CC}
I_{LI1}	D.C. Input Leakage Current to each pin of P0		+3	µA	V_{in} = 0 to V_{CC}
I_{IH}	Input High Current to $\overline{EA}$		100	µA	V_{IH} = 2.4V
I_{IL}	Input Low Current to each pin of P1, and to P2.6, P2.7.		−100	µA	V_{IL} = 0.45V
I_{IL1}	Input Low Current to $\overline{RESET}$	−0.3	−2	mA	V_{IL} = 0.45V
I_{IL2}	Input Low Current P2.2, P2.3, P2.4, READY, BUSWIDTH		−50	µA	V_{IL} = 0.45V
V_{OL}	Output Low Voltage on Quasi-Bidirectional port pins and P3, P4 when used as ports	0.45		V	I_{OL} = 0.8 mA (Note 1)
V_{OL1}	Output Low Voltage on Quasi-Bidirectional port pins and P3, P4 when used as ports	0.75		V	I_{OL} = 2.0 mA (Notes 1, 2, 3)
V_{OL2}	Output Low Voltage on Standard Output pins, $\overline{RESET}$ and Bus/Control Pins	0.45		V	I_{OL} = 0.8 mA (Note 1)
V_{OL3}	Output Low Voltage on Standard Output pins and $\overline{RESET}$	0.75		V	I_{OL} = 2.0 mA (Notes 1, 2, 3)

D.C. CHARACTERISTICS (Continued)

Symbol	Parameter	Min	Max	Units	Test Conditions
V_{OH}	Output High Voltage on Quasi-Bidirectional pins	2.4		V	$I_{OH} = -20\ \mu A$ (Note 1)
V_{OH1}	Output High Voltage on Standard Output pins and Bus/Control pins	2.4		V	$I_{OH} = -200\ \mu A$ (Note 1)
I_{OH3}	Output High Current on $\overline{RESET}$	−50		μA	$V_{OH} = 2.4V$
C_S	Pin Capacitance (Any Pin to V_{SS})		10	pF	fTEST = 1.0 MHz

NOTES:
1. Quasi-bidirectional pins include those on P1, for P2.6 and P2.7. Standard Output Pins include TXD, RXD (Mode 0 only), PWM, and HSO pins. Bus/Control pins include CLKOUT, ALE, $\overline{BHE}$, $\overline{RD}$, $\overline{WR}$, INST and AD0–15.
2. Maximum current per pin must be externally limited to the following values if V_{OL} is held above 0.45V.
   I_{OL} on quasi-bidirectional pins and Ports 3 and 4 when used as ports: 4.0 mA
   I_{OL} on standard output pins and $\overline{RESET}$: 8.0 mA
   I_{OL} on Bus/Control pins: 2.0 mA
3. During normal (non-transient) operation the following limits apply:
   Total I_{OL} on Port 1 must not exceed 8.0 mA.
   Total I_{OL} on P2.0, P2.6, $\overline{RESET}$ and all HSO pins must not exceed 15 mA.
   Total I_{OL} on Port 3 must not exceed 10 mA.
   Total I_{OL} on P2.5, P2.7, and Port 4 must not exceed 20 mA.

A.C. CHARACTERISTICS V_{CC}, V_{PD} = 4.5 to 5.5V; T_A = 0°C to 70°C; f_{OSC} = 6.0 to 12.0 MHz

Test Conditions: Load Capacitance on Output Pins = 80 pF
Oscillator Frequency = 12 MHz

TIMING REQUIREMENTS (Other system components must meet these specs.)

Symbol	Parameter	Min	Max	Units
T_{CLYX}(4)	READY Hold after CLKOUT Edge	0(1)		ns
T_{LLYV}	End of ALE/$\overline{ADV}$ to READY Valid		2Tosc−50	ns
T_{LLYH}	End of ALE/$\overline{ADV}$ to READY High	2Tosc+40	4Tosc−40	ns
T_{YLYH}	Non-Ready Time		1000	ns
T_{AVDV}(6)	Address Valid to Input Data Valid		5Tosc−90	ns
T_{RLDV}	$\overline{RD}$ Active to Input Data Valid		3Tosc−60	ns
T_{RHDX}	Data Hold after $\overline{RD}$ Inactive	0		ns
T_{RHDZ}	$\overline{RD}$ Inactive to Input Data Float	0	Tosc−20	ns
T_{AVGV}(4)(6)	Address Valid to BUSWIDTH Valid		2 Tosc −100	ns
T_{LLGX}(4)	BUSWIDTH Hold after ALE/$\overline{ADV}$ Low	Tosc +10		ns
T_{LLGV}(4)	ALE/$\overline{ADV}$ Low to BUSWIDTH Valid		Tosc −50	ns

NOTES:
1. If the 48-pin part is being used then this timing can be generated by assuming that the CLKOUT falling edge has occurred at 2Tosc+60 (TLLCH(max) + TCHCL(max)) after the falling edge of ALE.
4. Pins not bonded out on 48-pin parts.
6. The term "Address Valid" applies to AD0–15, $\overline{BHE}$ and INST.

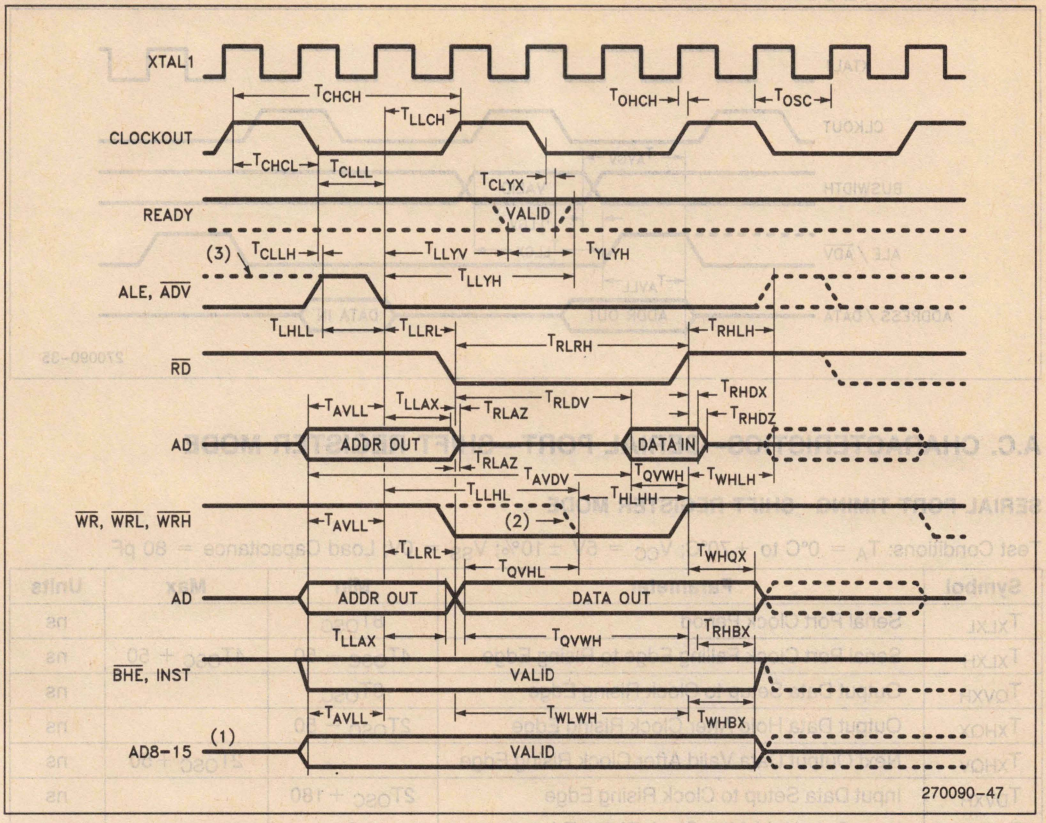
A.C. CHARACTERISTICS (Continued)

TIMING RESPONSES (MCS-96 parts meet these specs.)

Symbol	Parameter	Min	Max	Units
F_{XTAL}	Oscillator Frequency	6.0	12.0	MHz
T_{OSC}	Oscillator Period	83	166	ns
T_{OHCH}	XTAL1 Rising Edge to Clockout Rising Edge	0(4)	120(4)	ns
T_{CHCH}(4)	CLKOUT Period(3)	3Tosc(3)	3Tosc(3)	ns
T_{CHCL}(4)	CLKOUT High Time	Tosc−20	Tosc+20	ns
T_{CLLH}(4)	CLKOUT Low to ALE High	−5	20	ns
T_{LLCH}(4)	ALE/$\overline{ADV}$ Low to CLKOUT High	Tosc−20	Tosc+40	ns
T_{LHLL}	ALE/$\overline{ADV}$ High Time	Tosc−25	Tosc+15(5)	ns
T_{AVLL}(6)	Address Setup to End of ALE/$\overline{ADV}$	Tosc−50		ns
T_{RLAZ}(7)	$\overline{RD}$ or $\overline{WR}$ Low to Address Float		25	ns
T_{LLRL}	End of ALE/$\overline{ADV}$ to $\overline{RD}$ or $\overline{WR}$ Active	Tosc−20		ns
T_{LLAX}(7)	Address Hold after End of ALE/$\overline{ADV}$	Tosc−20		ns
T_{WLWH}	$\overline{WR}$ Pulse Width	3Tosc−35		ns
T_{QVWH}	Output Data Valid to End of $\overline{WR}$/$\overline{WRL}$/$\overline{WRH}$	3Tosc−60		ns
T_{WHQX}	Output Data Hold after $\overline{WR}$/$\overline{WRL}$/$\overline{WRH}$	Tosc−25		ns
T_{WHLH}	End of $\overline{WR}$/$\overline{WRL}$/$\overline{WRH}$ to ALE/$\overline{ADV}$ High	Tosc−55		ns
T_{RLRH}	$\overline{RD}$ Pulse Width	3Tosc−30		ns
T_{RHLH}	End of $\overline{RD}$ to ALE/$\overline{ADV}$ High	Tosc−25		ns
T_{CLLL}(4)	CLOCKOUT Low to ALE/$\overline{ADV}$ Low	Tosc−40	Tosc+20	ns
T_{RHBX}(4)	$\overline{RD}$ High to INST, $\overline{BHE}$, AD8-15 Inactive	Tosc	Tosc+30	ns
T_{WHBX}(4)	$\overline{WR}$ High to INST, $\overline{BHE}$, AD8-15 Inactive	Tosc−45	Tosc+100	ns
T_{HLHH}	$\overline{WRL}$, $\overline{WRH}$ Low to $\overline{WRL}$, $\overline{WRH}$ High	2Tosc−35	2Tosc+20	ns
T_{LLHL}	ALE/$\overline{ADV}$ Low to $\overline{WRL}$, $\overline{WRH}$ Low	2Tosc−20	2Tosc+55	ns
T_{QVHL}	Output Data Valid to $\overline{WRL}$, $\overline{WRH}$ Low	Tosc−60		ns

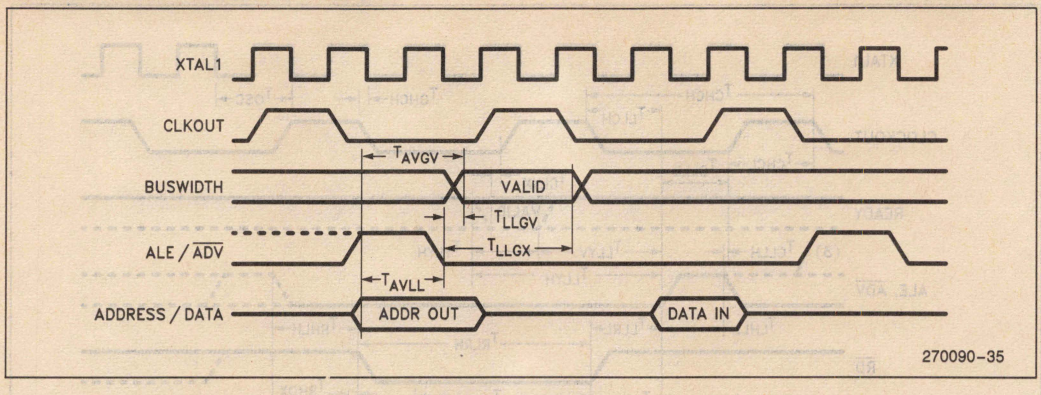
NOTES:
2. If more than one wait state is desired, add 3Tosc for each additional wait state.
3. CLKOUT is directly generated as a divide by 3 of the oscillator. The period will be 3Tosc ± 10 ns if Tosc is constant and the rise and fall times on XTAL1 are less than 10 ns.
4. Pins not bonded out on 48-pin parts.
5. Max spec applies only to ALE. Min spec applies to both ALE and $\overline{ADV}$.
6. The term "Address Valid" applies to AD0–15, $\overline{BHE}$ and INST.
7. The term " Address" in this definition applies to AD0–7 for 8-bit cycles, and AD0–15 for 16-bit cycles.

WAVEFORM



NOTES:
(1) 8-bit bus only.
(2) 8-bit bus; or when write strobe mode selected.
(3) When $\overline{ADV}$ selected.

WAVEFORM—BUSWIDTH PIN



A.C. CHARACTERISTICS—SERIAL PORT—SHIFT REGISTER MODE

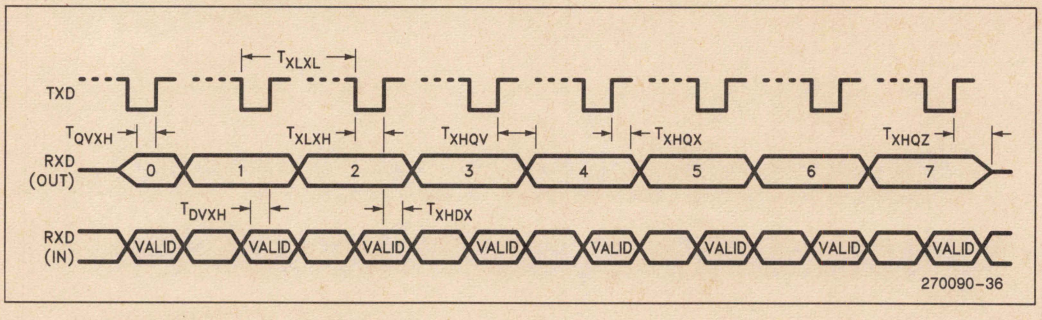
SERIAL PORT TIMING—SHIFT REGISTER MODE

Test Conditions: T_A = 0°C to +70°C; V_{CC} = 5V ±10%; V_{SS} = 0V; Load Capacitance = 80 pF

Symbol	Parameter	Min	Max	Units
T_{XLXL}	Serial Port Clock Period	$8T_{OSC}$		ns
T_{XLXH}	Serial Port Clock Falling Edge to Rising Edge	$4T_{OSC} - 50$	$4T_{OSC} + 50$	ns
T_{QVXH}	Output Data Setup to Clock Rising Edge	$3T_{OSC}$		ns
T_{XHQX}	Output Data Hold After Clock Rising Edge	$2T_{OSC} - 50$		ns
T_{XHQV}	Next Output Data Valid After Clock Rising Edge		$2T_{OSC} + 50$	ns
T_{DVXH}	Input Data Setup to Clock Rising Edge	$2T_{OSC} + 180$		ns
T_{XHDX}	Input Data Hold After Clock Rising Edge	0		ns
T_{XHQZ}	Last Clock Rising to Output Float		$5T_{OSC}$	ns

WAVEFORM—SERIAL PORT—SHIFT REGISTER MODE

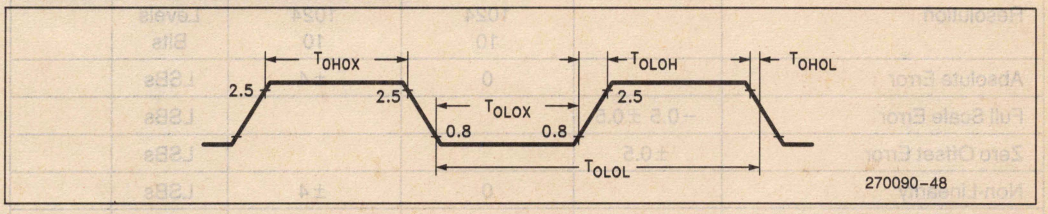
SERIAL PORT WAVEFORM—SHIFT REGISTER MODE



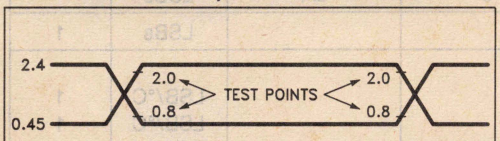
EXTERNAL CLOCK DRIVE

Symbol	Parameter	Min	Max	Units
$1/T_{OLOL}$	Oscillator Frequency	6	12	MHz
T_{OHOX}	High Time	25		ns
T_{OLOX}	Low Time	25		ns
T_{OLOH}	Rise Time		15	ns
T_{OHOL}	Fall Time		15	ns

EXTERNAL CLOCK DRIVE WAVEFORMS

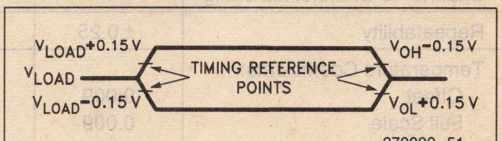


A.C. TESTING INPUT, OUTPUT WAVEFORM



A.C. Testing inputs are driven at 2.4V for a Logic "1" and 0.045V for a Logic "0". Timing measurements are made at 2.0V for a Logic "1" and 0.8V for a Logic "0".

FLOAT WAVEFORM



For Timing Purposes a Port Pin is no Longer Floating when a 100 mV change from Load Voltage Occurs, and Begins to Float when a 100 mV change from the Loaded V_{OH}/V_{OL} Level occurs $I_{OL}/I_{OH} \geq \pm 15$ mA.

A/D CONVERTER SPECIFICATIONS

A/D Converter operation is verified only on the 8097BH, 8397BH, 8095BH, 8395BH, 8797BH, 8795BH.

The absolute conversion accuracy is dependent on the accuracy of V_{REF}. The specifications given below assume adherence to the Operating Conditions section of these data sheets. Testing is done at V_{REF} = 5.120V.

OPERATING CONDITIONS

V_{CC}, V_{PD}, V_{REF} ..... 4.5V to 5.5V
V_{SS}, ANGND 0.0V
T_A ..... 0°C to 70°C
F_{OSC} ..... 6.0 to 12.0 MHz
Test Conditions:
V_{REF} 5.120V

Parameter	Typical*(1)	Minimum	Maximum	Units**	Notes
Resolution		1024 10	1024 10	Levels Bits	
Absolute Error		0	±4	LSBs	
Full Scale Error	−0.5 ±0.5			LSBs	
Zero Offset Error	±0.5			LSBs	
Non-Linearity		0	±4	LSBs	
Differential Non-Linearity		0	±2	LSBs	
Channel-to-Channel Matching		0	±1	LSBs	
Repeatability	±0.25			LSBs	1
Temperature Coefficients: Offset Full Scale Differential Non-Linearity	 0.009 0.009 0.009			 LSB/°C LSB/°C LSB/°C	 1 1 1
Off Isolation		−60		dB	1, 2, 4
Feedthrough	−60			dB	1, 2
V_{CC} Power Supply Rejection	−60			dB	1, 2
Input Resistance		1K	5K	Ω	1
D.C. Input Leakage		0	3.0	μA	
Sample Delay		$3T_{OSC} - 50$	$3T_{OSC} + 50$	ns	1, 3
Sample Time		$12T_{OSC} - 50$	$12T_{OSC} + 50$	ns	1

NOTES:
* These values are expected for most parts at 25°C.
** An "LSB", as used here, is defined in the glossary which follows and has a value of approximately 5 mV.
1. These values are not tested in production and are based on theoretical estimates and laboratory tests.
2. DC to 100 KHz.
3. For starting the A/D with an HSO Command.
4. Multiplexer Break-Before-Make Guaranteed.

A/D GLOSSARY OF TERMS

ABSOLUTE ERROR—The maximum difference between corresponding actual and ideal code transitions. Absolute Error accounts for all deviations of an actual converter from an ideal converter.

ACTUAL CHARACTERISTIC—The characteristic of an actual converter. The characteristic of a given converter may vary over temperature, supply voltage, and frequency conditions. An actual characteristic rarely has ideal first and last transition locations or ideal code widths. It may even vary over multiple conversions under the same conditions.

BREAK-BEFORE-MAKE—The property of a multiplexer which guarantees that a previously selected channel will be deselected before a new channel is selected. (e.g. the converter will not short inputs together.)

CHANNEL-TO-CHANNEL MATCHING—The difference between corresponding code transitions of actual characteristics taken from different channels under the same temperature, voltage and frequency conditions.

CHARACTERISTIC—A graph of input voltage versus the resultant output code for an A/D converter. It describes the transfer function of the A/D converter.

CODE—The digital value output by the converter.

CODE CENTER—The voltage corresponding to the midpoint between two adjacent code transitions.

CODE TRANSITION—The point at which the converter changes from an output code of Q, to a code of Q + 1. The input voltage corresponding to a code transition is defined to be that voltage which is equally likely to produce either of two adjacent codes.

CODE WIDTH—The voltage corresponding to the difference between two adjacent code transitions.

CROSSTALK—See "Off-Isolation".

D.C. INPUT LEAKAGE—Leakage current to ground from an analog input pin.

DIFFERENTIAL NON-LINEARITY—The difference between the ideal and actual code widths of the terminal based characteristic.

FEEDTHROUGH—Attenuation of a voltage applied on the selected channel of the A/D Converter after the sample window closes.

FULL SCALE ERROR—The difference between the expected and actual input voltage corresponding to the full scale code transition.

IDEAL CHARACTERISTIC—A characteristic with its first code transition at V_{IN} = 0.5 LSB, its last code transition at V_{IN} = (V_{REF} − 1.5 LSB) and all code widths equal to one LSB.

INPUT RESISTANCE—The effective series resistance from the analog input pin to the sample capacitor.

LSB—Least Significant Bit: The voltage corresponding to the full scale voltage divided by 2^n, where n is the number of bits of resolution of the converter. For a 10-bit converter with a reference voltage of 5.12V, one LSB is 5.0 mV. Note that this is different than digital LSBs, since an uncertainty of two LSB, when referring to an A/D converter, equals 10 mV. (This has been confused with an uncertainty of two digital bits, which would mean four counts, or 20 mV.)

MONOTONIC—The property of successive approximation converters which guarantees that increasing input voltages produce adjacent codes of increasing value, and that decreasing input voltages produce adjacent codes of decreasing value.

NO MISSED CODES—For each and every output code, there exists a unique input voltage range which produces that code only.

NON-LINEARITY—The maximum deviation of code transitions of the actual characteristic from the corresponding code transitions of the terminal based characteristic of the converter.

OFF-ISOLATION—Attenuation of a voltage applied on a deselected channel of the A/D converter. (Also referred to as Crosstalk.)

REPEATABILITY—The difference between corresponding code transitions from different actual characteristics taken from the same converter on the same channel at the same temperature, voltage and frequency conditions.

RESOLUTION—The number of input voltage levels that the converter can unambiguously distinguish between. Also defines the number of useful bits of information which the converter can return.

SAMPLE DELAY—The delay from receiving the start conversion signal to when the sample window opens.

SAMPLE DELAY UNCERTAINTY—The variation in the sample delay.

SAMPLE TIME—The time that the sample window is open.

SAMPLE TIME UNCERTAINTY—The variation in the sample time.

SAMPLE WINDOW—Begins when the sample capacitor is attached to a selected channel and ends when the sample capacitor is disconnected from the selected channel.

SUCCESSIVE APPROXIMATION—An A/D conversion method which uses a binary search to arrive at the best digital representation of an analog input.

TEMPERATURE COEFFICIENTS—Change in the stated variable per degree centigrade temperature change. Temperature coefficients are added to the typical values of a specification to see the effect of temperature drift.

TERMINAL BASED CHARACTERISTIC—An actual characteristic which has been rotated and translated to remove zero offset and full scale error.

V_{CC} REJECTION—Attenuation of noise on the V_{CC} line to the A/D converter.

ZERO OFFSET—The difference between the expected and actual input voltage corresponding to the first code transition.

EPROM CHARACTERISTICS

The 879XBH contains 8K bytes of ultraviolet Erasable and Electrically Programmable Read Only Memory (EPROM) for internal storage. This memory can be programmed in a variety of ways—including at run-time under software control.

The EPROM is mapped into memory locations 2000H through 3FFFH if $\overline{EA}$ is a TTL high. However, applying +12.75V to $\overline{EA}$ when the chip is reset will place the 879XBH in EPROM Programming Mode. The Programming Mode has been implemented to support EPROM programming and verification.

When an 879XBH is in Programming Mode, special hardware functions are available to the user. These functions include algorithms for slave, gang and auto EPROM programming.

Programming the 879XBH

Three flexible EPROM programming modes are available on the 879XBH—auto, slave and run-time. These modes can be used to program 879XBHs in a gang, stand alone or run-time environment.

The Auto Programming Mode enables an 879XBH to program itself, and up to 15 other 879XBHs, with the 8K bytes of code beginning at address 4000H on its external bus. The Slave Mode provides a standard interface that enables any number of 879XBHs to be programmed by a master device such as an EPROM programmer. The Run-Time Mode allows individual EPROM locations to be programmed at run-time under complete software control.

In the Programming Mode, some I/O pins have been renamed. These new pin functions are used to determine the programming function that is performed, provide programming ALEs, provide slave ID numbers and pass error information. Figure 19 shows how the pins are renamed. Figure 20 describes each new pin function.

While in Programming Mode, PMODE selects the programming function that is performed (see Figure 18). When not in the Programming Mode, Run-Time programming can be done at any time.

PMODE	Programming Mode
0–4	Reserved
5	Slave Programming
6–0BH	Reserved
0CH	Auto Programming Mode
0DH	Program Configuration Byte
0EH–0FH	Reserved

Figure 18. Programming Function PMODE Values

To guarantee proper execution, the pins of PMODE and SID must be in their desired state before the $\overline{RESET}$ pin is allowed to rise and reset the part. Once the part is reset, it is in the selected mode and should not be switched to another mode without a new reset sequence.

When $\overline{EA}$ selects the Programming Mode, the chip reset sequence loads the CCR from the Programming Chip Configuration Byte (PCCB). This is a separate EPROM location that is not mapped under normal operation. PCCR is only important when programming in the Auto Programming Mode. In this mode, the 879XBH that is being programmed gets the data to be programmed from external memory over the system bus. Therefore, PCCR must correctly correspond to the memory system in the programming setup, which is not necessarily the memory organization of the application.

The following sections describe 879XBH programming in each programming mode.

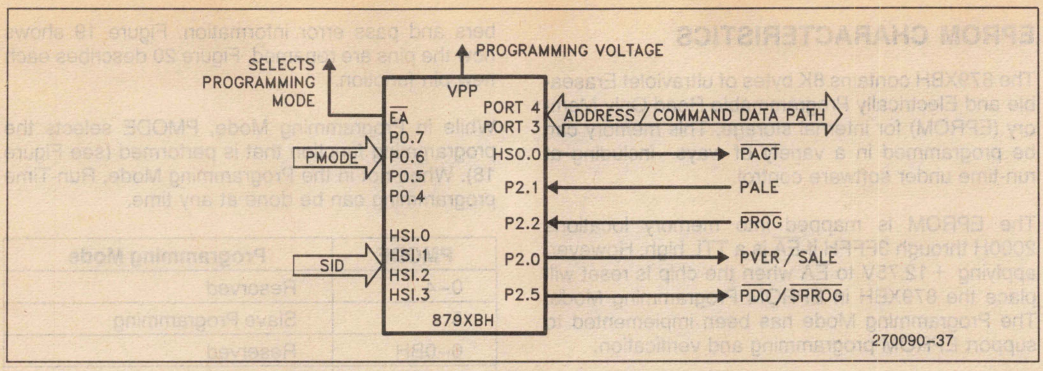


Figure 19. Programming Mode Pin Functions

Name	Function
PMODE	Programming Mode Select. Determines the EPROM programming algorithm that is performed. PMODE is sampled after a chip reset and should be static while the part is operating.
SID	Slave ID Number. Used to assign each slave a pin of Port 3 or 4 to use for passing programming verification acknowledgement. For example, if gang programming in the Slave Programming Mode, the slave with SID = 0001 will use Port 3.1 to signal correct or incorrect program verification.
PALE	Programming ALE input. Accepted by an 879XBH that is in the Slave Programming Mode. Used to indicate that Ports 3 and 4 contain a command/address.
$\overline{\text{PROG}}$	Programming Pulse. Accepted by an 879XBH that is in the Slave Programming Mode. Used to indicate that Ports 3 and 4 contain the data to be programmed. A falling edge on $\overline{\text{PROG}}$ signifies data valid and starts the programming cycle. A rising edge on $\overline{\text{PROG}}$ will halt programming in the slaves.
$\overline{\text{PACT}}$	Programming Active. Used in the Auto Programming Mode to indicate when programming activity is complete.
PVER	Program Verified. A signal output after a programming operation by parts in the Slave Programming Mode.
$\overline{\text{PDO}}$	Programming Duration Overflowed. A signal output by parts in the Slave Programming Mode. Used to signify that the $\overline{\text{PROG}}$ pulse applied for a programming operation was longer than allowed.
SALE	Slave ALE. Output signal from an 879XBH in the Auto Programming Mode. A falling edge on SALE indicates that Ports 3 and 4 contain valid address/command information for slave 879XBHs that may be attached to the master.
$\overline{\text{SPROG}}$	Slave Programming Pulse. Output from an 879XBH in the Auto Programming Mode. A falling edge on $\overline{\text{SPROG}}$ indicates that Ports 3 and 4 contain valid data for programming into slave 879XBHs that may be attached to the master.
PORTS 3 and 4	Address/Command/Data Bus. Used to pass commands, addresses and data to and from slave mode 879XBHs. Used by chips in the Auto Programming Mode to pass command, addresses and data to slaves. Also used in the Auto Programming Mode as a regular system bus to access external memory. Should have pullups to V_{CC} (15 KΩ).

Figure 20. Programming Mode Pin Definitions

AUTO PROGRAMMING MODE

The Auto Programming Mode provides the ability to program the internal 879XBH EPROM without having to use a special EPROM programmer. In this mode, the 879XBH simply programs itself with the data found at external locations 4000H through 5FFFH. All that is required is that some sort of external memory reside at these locations, that $\overline{EA}$ selects the Programming Mode and that V_{PP} is applied. Figure 21 shows a minimum configuration for using an 8K × 8 EPROM to program one 879XBH in the Auto Programming Mode.

The 879XBH first reads a word from external memory, then the Modified Quick-Pulse Programming™ Algorithm (described later) is used to program the appropriate EPROM location. Since the erased state of a byte is 0FFH, the Auto Programming Mode will skip locations where the data to be programmed is 0FFH. When all 8K has been programmed, $\overline{PACT}$ goes high and the part outputs a 0 on Port 3.0 if it programmed correctly and a 1 if it failed.

Gang Programming with the Auto Programming Mode

An 879XBH in the Auto Programming Mode can also be used as a programmer for up to 15 other 879XBHs that are configured in the Slave Programming Mode. To accomplish this, the 879XBH acting as the master outputs the slave command/data pairs on Ports 3 and 4 necessary to program slave parts with the same data it is programming itself with. Slave ALE (SALE) and Slave $\overline{PROG}$ ($\overline{SPROG}$) signals are provided by the master to the slaves to demultiplex the commands from the data. Figure 22 is a block diagram of a gang programming system using one 879XBH in the Auto Programming Mode. The Slave Programming Mode is described in the next section.

The master 879XBH first reads a word from the external memory controlled by ALE, $\overline{RD}$ and $\overline{WR}$. It then drives Ports 3 and 4 with a Data Program command using the appropriate address and alerts the slaves with a falling edge on SALE. Next, the data to be programmed is driven onto Ports 3 and 4 and slave programming begins with a falling edge on $\overline{SPROG}$. At the same time, the master begins to program its own EPROM location with the data read in. Intel's Modified Quick-Pulse Programming™ Algorithm is used, with Data Verify commands being given to the slaves after each programming pulse.

When programming is complete, $\overline{PACT}$ goes high and Ports 3 and 4 are driven with all 1s if all parts programmed correctly. Individual bits of Port 3 and 4 will be driven to 0 if the slave with that bit number as an SID did not program correctly. The 879XBH used as the master assigns itself an SID of 0.

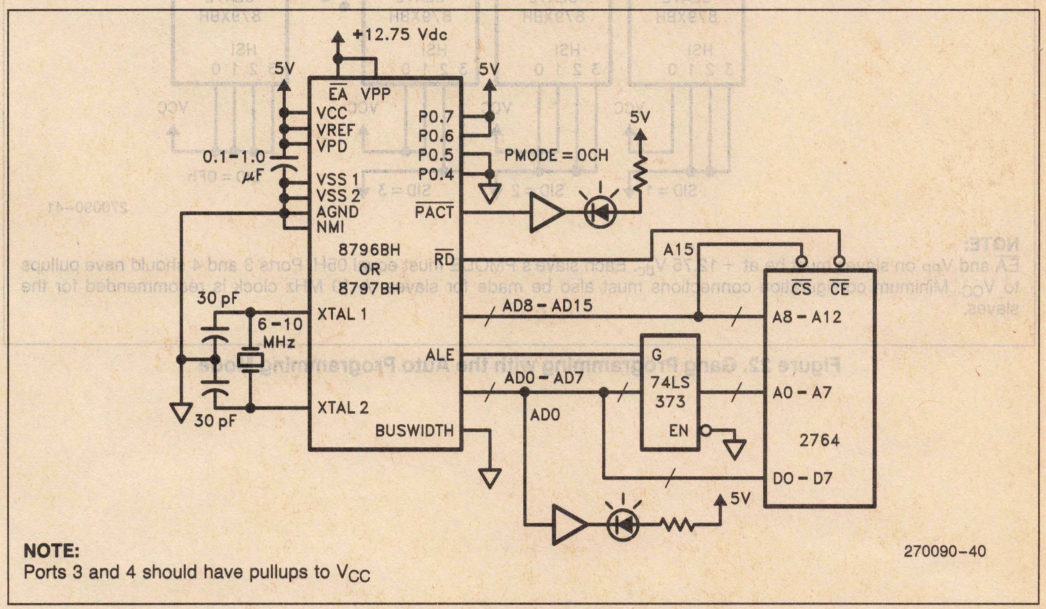
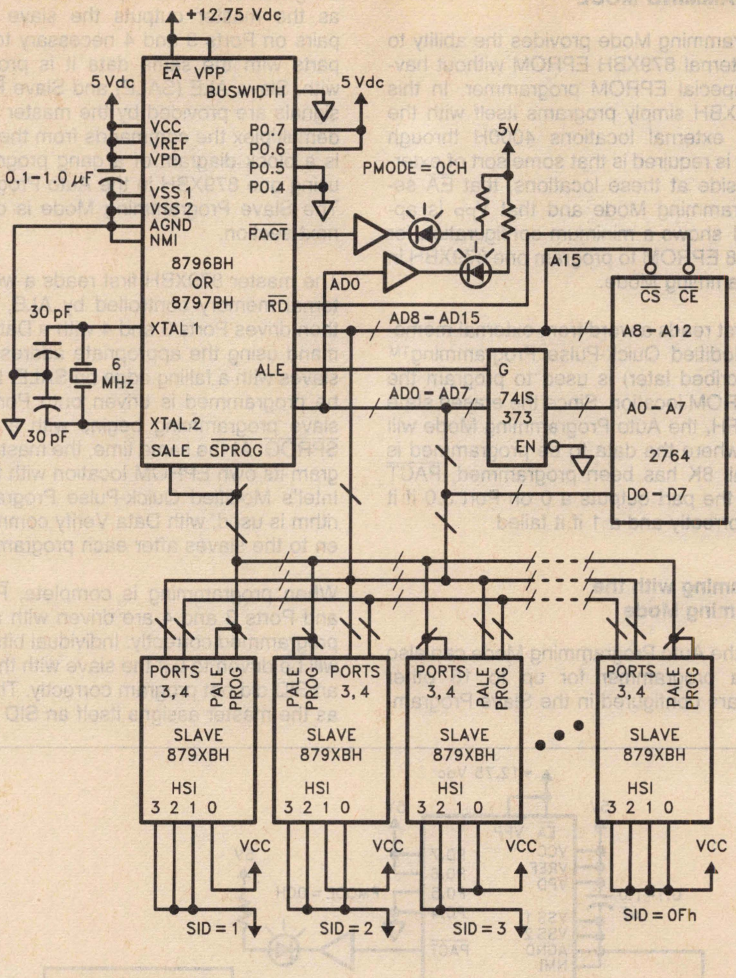


Figure 21. The Auto Programming Mode



NOTE:
$\overline{EA}$ and V_{PP} on slaves must be at $+12.75\ V_{dc}$. Each slave's PMODE must equal 05H. Ports 3 and 4 should have pullups to V_{CC}. Minimum configuration connections must also be made for slaves. A 10 MHz clock is recommended for the slaves.

Figure 22. Gang Programming with the Auto Programming Mode

SLAVE PROGRAMMING MODE

Any number of 879XBHs can be programmed by a master programmer through the Slave Programming Mode.

The programming device uses Ports 3 and 4 of the parts being programmed as a command/data path. The slaves accept signals on PALE (Program ALE) and $\overline{\text{PROG}}$ (Program Enable) to demultiplex the commands and data. The slaves also use PVER, $\overline{\text{PDO}}$ and Ports 3 and 4 to pass error information to the programmer. Support for gang programming of up to 16 879XBHs is provided. If each part is given a unique SID (Slave ID Number) an 879XBH in the Auto Programming Mode can be used as a master to program itself and up to 15 other slave 879XBHs. There is, however, no 879XBH dependent limit to the number of parts that can be gang programmed in the slave mode.

It is important to note that the interface to an 879XBH in the slave mode is similar to a multiplexed bus. Attempting to issue consecutive PALE pulses without a corresponding $\overline{\text{PROG}}$ pulse will produce unexpected results. Similarly, issuing consecutive $\overline{\text{PROG}}$ pulses without the corresponding PALE pulses immediately preceding is equally unpredictable.

Slave Programming Commands

The commands sent to the slaves are 16-bits wide and contain two fields. Bits 14 and 15 specify the action that the slaves are to perform. Bits 0 through 13 specify the address upon which the action is to take place. Commands are sent via Ports 3 and 4 and are available to cause the slaves to program a word, verify a word, or dump a word (Table 4). The address part of the command sent to the slaves ranges from 2000H to 3FFFH and refers to the internal EPROM memory space. The following sections describe each Slave Programming Mode command.

Table 4. Slave Programming Mode Commands

P4.7	P4.6	Action
0	0	Word Dump
0	1	Data Verify
1	0	Data Program
1	1	Reserved

DATA PROGRAM COMMAND—After a Data Program Command has been sent to the slaves, $\overline{\text{PROG}}$ must be pulled low to cause the data on Ports 3 and 4 to be programmed into the location specified during the command. The falling edge of $\overline{\text{PROG}}$ is not only used to indicate data valid, but also triggers the hardware programming of the word specified. The slaves will begin programming 48 states after $\overline{\text{PROG}}$ falls, and will continue to program the location until $\overline{\text{PROG}}$ rises.

After the rising edge of $\overline{\text{PROG}}$, the slaves automatically perform a verification of the address just programmed. The result of this verification is then output on PVER (Program Verify) and $\overline{\text{PDO}}$ (Program Duration Overflowed). Therefore, verification information is available following the Data Program Command for programming systems that cannot use the Data Verify command.

If PVER and $\overline{\text{PDO}}$ of all slaves are 1s after $\overline{\text{PROG}}$ rises then the data program was successful everywhere. If PVER is a 0 in any slave, then the data programmed did not verify correctly in that part. If $\overline{\text{PDO}}$ is a 0 in any slave, then the programming pulse in those parts was terminated by an internal safety feature rather than the rising edge of $\overline{\text{PROG}}$. The safety feature prevents over-programming in the slave mode. Figure 23 shows the relationship of PALE, $\overline{\text{PROG}}$, PVER and $\overline{\text{PDO}}$ to the Command/Data Path on Ports 3 and 4 for the Data Program Command.

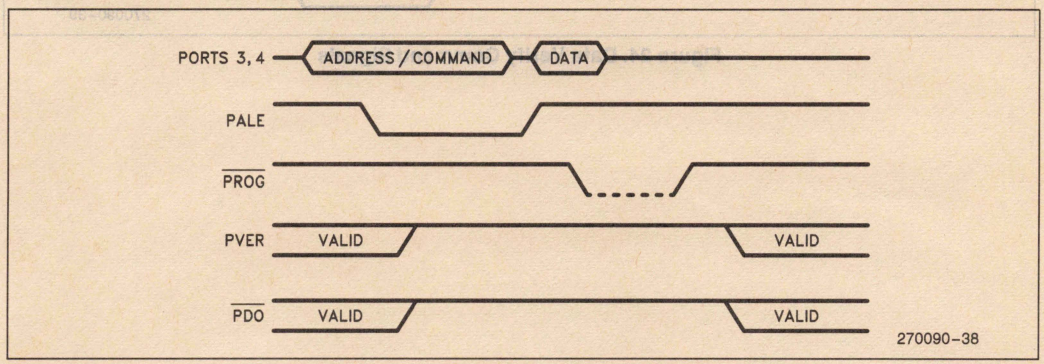


Figure 23. Data Program Signals in Slave Programming Mode

DATA VERIFY COMMAND—When the Data Verify Command is sent, the slaves respond by driving one bit of Port 3 or 4 to indicate correct or incorrect verification of the previous Data Program. A 1 indicates correct verification, while a 0 indicates incorrect verification. The SID (Slave ID Number) of each slave determines which bit of the command/data path is driven. $\overline{\text{PROG}}$ from the programmer governs when the slaves drive the bus. Figure 24 shows the relationship of Ports 3 and 4 to PALE and $\overline{\text{PROG}}$.

This command is always preceded by a Data Program Command in a programming system with as many as 16 slaves. However, a Data Verify Command does not have to follow every Data Program Command.

WORD DUMP COMMAND — When the Word Dump Command is issued, the 879XBH being programmed adds 2000H to the address field of the command and places the value found at the new address on Ports 3 and 4. For example, sending the command #0100H to a slave will result in the slave placing the word found at location 2100H on Ports 3 and 4. $\overline{\text{PROG}}$ from the programmer governs when the slave drives the bus. The signals are the same as shown in Figure 24.

Note that this command will work only when just one slave is attached to the bus, and that there is no restriction on commands that precede or follow a Word Dump Command.

Gang Programming with the Slave Programming Mode

Gang programming of 879XBHs can be done using the Slave Programming Mode. There is no 879XBH based limit on the number of chips that may be hooked to the same Port 3/Port 4 data path for gang programming.

If more than 16 chips are being gang programmed, the PVER and $\overline{\text{PDO}}$ outputs of each chip could be used for verification. The master programmer could issue a data program command then either watch every chip's error signals, or AND all the signals together to get a system PVER and $\overline{\text{PDO}}$.

If 16 or fewer 879XBHs are to be gang programmed at once, a more flexible form of verification is available. By giving each chip being programmed a unique SID, the master programmer could then issue a data verify command after the data program command. When a verify command is seen by the slaves, each will drive one pin of Port 3 or 4 with a 1 if the programming verified correctly or a 0 if programming failed. The SID is used by each slave to determine which Port 3, 4 bit it is assigned. An 879XBH in the auto programming mode could be the master programmer if 15 or fewer slaves need to be programmed (See Gang Programming with the Auto Programming Mode).

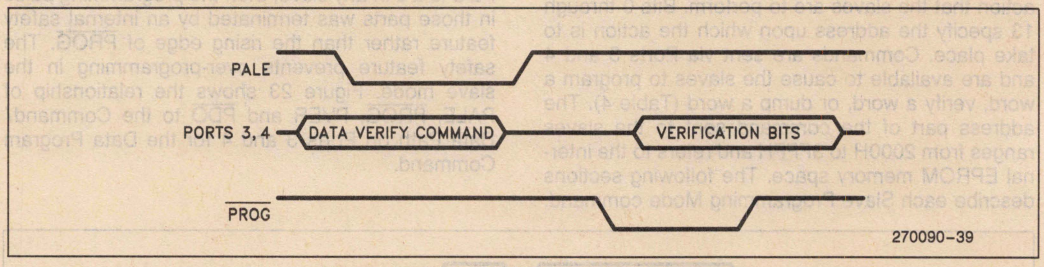


Figure 24. Data Verify Command Signals

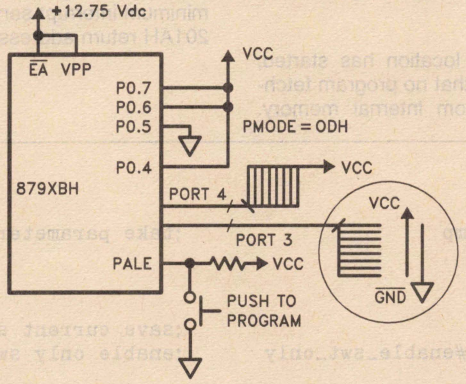
AUTO CONFIGURATION BYTE PROGRAMMING MODE

The CCB (location 2018H) can be treated just like any other EPROM location, and programmed using any programming mode. But to provide for simple programming of the CCB when no other locations need to be programmed, the Auto Configuration Byte Programming Mode is provided. Programming in this mode also programs PCCB. Figure 25 shows a block diagram for using the Auto Configuration Byte Programming Mode.

With PMODE = 0DH and OFF on Port 4, CCB and PCCB will be programmed the value on Port 3 when a logic 0 is placed on PALE. After programming is complete, PVER will be driven to a 1 if the bytes programmed correctly, and a 0 if the programming failed.

This method of programming is the only way to program PCCB. PCCB is a non-memory mapped EPROM location that gets loaded into CCR during the reset sequence when the voltage on EA puts the 879XBH in Programming Mode. If PCCB is not programmed using the Auto Configuration Byte Programming Mode, every time the 879XBH is put into Programming Mode the CCR will be loaded with 0FFH (the value of the erased PCCB location).

However, if programming of the CCB and PCCB is done using this programming mode, the PCCB will take on the value programmed into CCB. This means that until the part is erased, programming activities that use the system will employ the bus width and controls selected by the user's CCB.



270090-42

NOTE:
Tie Port 3 to the value desired to be programmed into CRB, and PCCB. Make all necessary minimum connections for power, ground and clock.

Figure 25. The Auto CCR Programming Mode

RUN-TIME PROGRAMMING

Run-Time Programming of the 879XBH is provided to allow the user complete flexibility in the ways in which the internal EPROM is programmed. That flexibility includes the ability to program just one byte or one word instead of the whole EPROM, and extends to the hardware necessary to program. The only additional requirement of a system is that a programming voltage is applied to V_{PP}. Run-Time Programming is done with $\overline{EA}$ at TTL-high (normal operation - internal/external access).

To Run-Time program, the user writes a byte or word to the location to be programmed. Once this is done, the 879XBH will continue to program that location until another data read from or data write to the EPROM occurs. The user can therefore control the duration of the programming pulse to within a few microseconds. An inteligent algorithm should be implemented in software. It is recommended that the Modified Quick-Pulse Programming™ Algorithm be implemented.

After the programming of a location has started, care must be taken to insure that no program fetches (or pre-fetches) occur from internal memory. This is of no concern if the program is executing from external memory. However, if the program is executing from internal memory when the write occurs, it will be necessary to use the built in "Jump to Self" located at 201AH.

"Jump to Self" is a two byte instruction in the Intel test ROM which can be CALLed after the user has started programming a location by writing to it. A software timer interrupt could then be used to escape from the "Jump to Self" when the proper programming pulse duration has elapsed. Figure 26 is an example of how to program an EPROM location while execution is entirely internal.

Upon entering the PROGRAM routine, the address and data are retrieved from the STACK and a Software Timer is set to expire one programming pulse later. The data is then written to the EPROM location and a CALL to location 201AH is made. Location 201AH is in Intel reserved test ROM, and contains the two byte opcode for a "Jump to Self." The minimum interrupt service routine would remove the 201AH return address from the STACK and return.

```
PROGRAM:
     POP   temp
     POP   address_temp                  ;take parameters from the STACK
     POP   data--temp
     PUSH  temp

     PUSHF                               ;save current status
     LDB   int_mask ,#enable_swt_only    ;enable only swt interrupts

     LDB   HSO_COMMAND ,#SWT0_ovf        ;load swt command to interrupt
     ADD   HSO_TIME,TIMER1,#program_pulse ;when program pulse time
                                         ;has elasped

     EI
     ST    data_temp, [address_temp]
     CALL  201AH

     POPF
     RET

SWT_ISR:
     . . .

swt0_expired:
     POP   0
     RET
     . . .
```

Figure 26. Programming the EPROM from Internal Memory Execution

ROM/EPROM PROGRAM LOCK

Protection mechanisms have been provided on the ROM and EPROM versions of the 809XBH to inhibit unauthorized accesses of internal program memory. However, there must always be a way to allow authorized program memory dumps for testing purposes. The following describes 839XBH, 879XBH program lock features and the mode provided for authorized memory dumps.

PROGRAM LOCK FEATURES

Write protection is provided for EPROM parts, while READ protection is provided for both ROM and EPROM parts.

Write protection is enabled by causing the LOC0 bit in the CCR to take the value 0. When WRITE protection is selected, the bus controller will cycle through the write sequence, but will not actually drive data to the EPROM and will not enable V_{PP} to the EPROM. This protects the entire EPROM 2000H-3FFFH from inadvertant or unauthorized programming.

READ protection is selected by causing the LOC1 bit in the CCR to take the value 0. When READ protection is enabled, the bus controller will only perform a data read from the address range 2020H-3FFFH if the slave program counter is in the range 2000H-3FFFH. Note that since the slave PC can be many bytes ahead of the CPU program counter, an instruction that is located after address 3FFAH may not be allowed to access protected memory, even through the instruction is itself protected.

If the bus controller receives a request to perform a READ of protected memory, the READ sequence occurs with indeterminant data being returned to the CPU.

Other enhancements were also made to the 8096BH for program protection. For example, the value of $\overline{EA}$ is latched on reset so that the device cannot be switched from external to internal execution mode at run-time. In addition, if READ protection is selected, an NMI event will cause the device to switch to external only execution mode. Internal execution can only resume by resetting the chip.

AUTHORIZED ACCESS OF PROTECTED MEMORY

To provide a method of dumping the internal ROM/EPROM for testing purposes a "Security Key" mechanism and ROM dump mode have been implemented.

The security key is a 128 bit number, located in internal memory, that must be matched before a ROM dump will occur. The application code contains the security key starting at location 2020H.

The ROM dump mode is entered just like any programming mode ($\overline{EA}$ = 12.75V), except that a special PMODE strapping is used. The PMODE for ROM dump is 6H (0110b).

The ROM dump sequence begins with a security key verification. Users must place at external locations 4020H–402FH the same 16 byte key that resides inside the chip at locations 2020H-202FH. Before doing a ROM dump, the chip checks that the keys match.

After a successful key verification, the chip dumps data to external locations 1000H-11FFH and 4000H-5FFFH. Unspecified data appears at the low addresses. Internal EPROM/ROM is dumped to 4000H-5FFFH beginning with internal address 2000H.

If a security key verification is not successful, the chip will put itself into an endless loop of internal execution.

NOTE:

Substantial effort has been expended to provide an excellent program protection scheme. However, Intel cannot, and does not guarantee that the protection methods that we have devised will prevent unauthorized access.

MODIFIED QUICK-PULSE PROGRAMMING™ ALGORITHM

The Modified Quick-Pulse Programming™ Algorithm calls for each EPROM location to receive 25 separate 100 μs ($\pm$5 μs) programming cycles. Verification of correct programming is done after the 25 pulses. If the location verifies correctly, the next location is programmed. If the location fails to verify, the location has failed.

Once all locations are programmed and verified, the entire EPROM is again verified.

Programming of 879XBH parts is done with V_{PP} = 12.75V $\pm$0.25V and V_{CC} = 5.0V $\pm$0.5V.

SIGNATURE WORD

The 879XBH contains a signature word at location 2070H. The word can be accessed in the slave mode by executing a word dump command.

Table 5. 8X9XBH Signature Word

Device	Signature Word
879XBH	896FH
839XBH	896EH
809XBH	Undefined

Erasing the 879XBH EPROM

Initially, and after each erasure, all bits of the 879XBH are in the "1" state. Data is introduced by selectively programming "0s" into the desired bit locations. Although only "0s" will be programmed, both "1s" and "0s" can be present in the data word. The only way to change a "0" to a "1" is by ultraviolet light erasure.

The erasure characteristics of the 879XBH are such that erasure begins to occur upon exposure to light with wavelengths shorter than approximately 4000 Angstroms (Å). It should be noted that sunlight and certain types of fluorescent lamps have wavelengths in the 3000–4000 Å range. Constant exposure to room level fluorescent lighting could erase the typical 879XBH in approximately 3 years, while it would take approximately 1 week to cause erasure when exposed to direct sunlight. If the 879XBH is to be exposed to light for extended periods of time, opaque labels must be placed over the EPROM's window to prevent unintentional erasure.

The recommended erasure procedure for the 879XBH is exposure to shortwave ultraviolet light which has a wavelength of 2537 Å. The integrated dose (i.e., UV intensity × exposure time) for erasure should be a minimum of 15 Wsec/cm^2. The erasure time with this dosage is approximately 15 to 20 minutes using an ultraviolet lamp with a 12000 μW/cm^2 power rating. The 879XBH should be placed within 1 inch of the lamp tubes during erasure. The maximum integrated dose an 879XBH can be exposed to without damage is 7258 Wsec/cm^2 (1 week @ 12000 μW/cm^2). Exposure of the 879XBH to high intensity UV light for long periods may cause permanent damage.

POWER SUPPLY SEQUENCE WHILE PROGRAMMING

For any 879XBH that is in any programming mode, high voltages must be applied to the device. To avoid damaging the parts, the following rules must not be violated.

RULE #1— V_{PP} must not have a low impedance path to ground when V_{CC} is above 4.5V.

RULE #2— V_{CC} must be above 4.5V before V_{PP} can be higher than 5.0V.

RULE #3— V_{PP} must be within 1V of V_{CC} while V_{CC} is below 4.5V.

RULE #4— All voltages must be within tolerance and the oscillator stable before $\overline{RESET}$ rises.

To adhere to these rules, the following power up and power down sequences can be followed.

POWER UP

$\overline{RESET}$ = 0;
CLOCK ON; if using an external clock
; instead of an oscillator
$V_{CC} = V_{PP} = V_{EA}$ = 5V;
PALE = $\overline{PROG}$ = PORT 34 = V_{IH};*
SID AND PMODE VALID;
$V_{EA} = V_{PP}$ = 12.75V;
WAIT; wait for supplies and clock to
; settle
$\overline{RESET}$ = 5V;
WAIT Tshll; See Data Sheet
BEGIN;

POWER DOWN

$\overline{RESET}$ = 0;
$V_{PP} = V_{EA}$ = 5V;
PALE = $\overline{PROG}$ = SID = PMODE = PORT34 = 0V;
$V_{CC} = V_{PP} = V_{EA}$ = 0V;
CLOCK OFF;

*V_{IH} = Logical "1", 2.4V Minimum

One final note on power up, power down. The maximum limit on V_{PP} must never be violated, even for an instant. Therefore, an RC rise to the desired V_{PP} is recommended. V_{PP} is also sensitive to instantaneous voltage steps. This also can be avoided by using an RC ramp on V_{PP}.

EPROM SPECIFICATIONS

A.C. EPROM PROGRAMMING CHARACTERISTICS

Operating Conditions: Load Capacitance = 150 pF, T_A = 25°C ±5°C, V_{CC}, V_{PD}, V_{REF} = 5.0V ± 0.5V, V_{SS}, AGND = 0V, V_{PP} = 12.75V ± 0.25V, $\overline{EA}$ = 11V ± 2.0V, f_{osc} = 6.0 MHz

Symbol	Parameter	Min	Max	Units
T_{AVLL}	ADDRESS/COMMAND Valid to PALE Low	0		T_{osc}
T_{LLAX}	ADDRESS/COMMAND Hold After PALE Low	80		T_{osc}
T_{DVPL}	Output Data Setup Before $\overline{PROG}$ Low	0		T_{osc}
T_{PLDX}	Data Hold After $\overline{PROG}$ Falling	80		T_{osc}
T_{LLLH}	PALE Pulse Width	180		T_{osc}
T_{PLPH}	$\overline{PROG}$ Pulse Width	250 T_{osc}	100 µS + 144 T_{osc}	
T_{LHPL}	PALE High to $\overline{PROG}$ Low	250		T_{osc}
T_{PHLL}	$\overline{PROG}$ High to Next PALE Low	600		T_{osc}
T_{PHDX}	Data Hold After $\overline{PROG}$ High	30		T_{osc}
T_{PHVV}	$\overline{PROG}$ High to PVER/$\overline{PDO}$ Valid	500		T_{osc}
T_{LLVH}	PALE Low to PVER/$\overline{PDO}$ High	100		T_{osc}
T_{PLDV}	$\overline{PROG}$ Low to VERIFICATION/DUMP Data Valid	100		T_{osc}
T_{SHLL}	RESET High to First PALE Low (not shown)	2000		T_{osc}

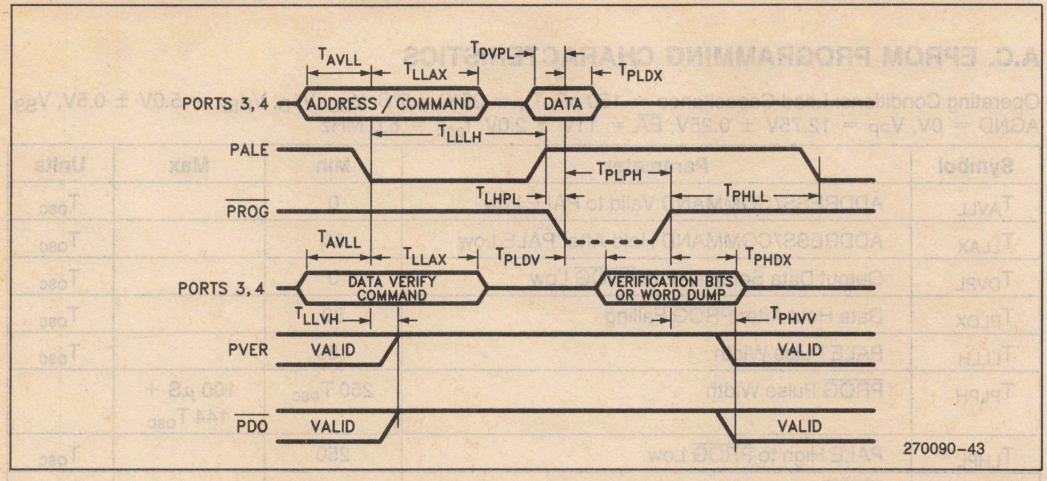
NOTE:
Run-time programming is done with F_{osc} = 6.0 MHz to 12.0 MHz, V_{CC}, V_{PD}, V_{REF} = 5V ± 0.5V, T_A = 25°C to ±5°C and V_{PP} = 12.75V ± 0.25V. For run-time programming over a full operating range, contact the factory.

D.C. EPROM PROGRAMMING CHARACTERISTICS

Symbol	Parameter	Min	Max	Units
I_{PP}	V_{PP} Supply Current (Whenever Programming)		100	mA

NOTE:
V_{PP} must be within 1V of V_{CC} while V_{CC} < 4.5V. V_{PP} must not have a low impedance path to ground or V_{SS} while V_{CC} > 4.5V.

WAVEFORM—EPROM PROGRAMMING



Reserved location warning: Intel Reserved addresses can not be used by applications which use 8X9XBH internal ROM/EPROM. The data read from a reserved location is not guaranteed, and a write to any reserved location could cause unpredictable results. When attempting to program Intel Reserved addresses, the data must be 0FFFFH to ensure a harmless result. A memory map indicating reserved locations on the 8X9XBH is shown in Figure 27.

Intel Reserved locations, when mapped to external memory, must be filled with 0FFFFH to ensure compatibility with future parts.

EXTERNAL MEMORY OR I/O	FFFFH 4000H
INTERNAL PROGRAM STORAGE ROM/EPROM OR EXTERNAL MEMORY	2080H
RESERVED	2072H–207FH
SIGNATURE WORD	2070H–2071H
RESERVED	2030H–206FH
SECURITY KEY	2020H–202FH
RESERVED	201CH–201FH
SELF JUMP CODE (27 FE)	201AH–201BH
RESERVED	2019H
CHIP CONFIGURATION BYTE	2018H
RESERVED	2012H-2017H
INTERRUPT VECTORS	2000H

Figure 27. Reserved Locations

MCS®-96 Application Notes 16

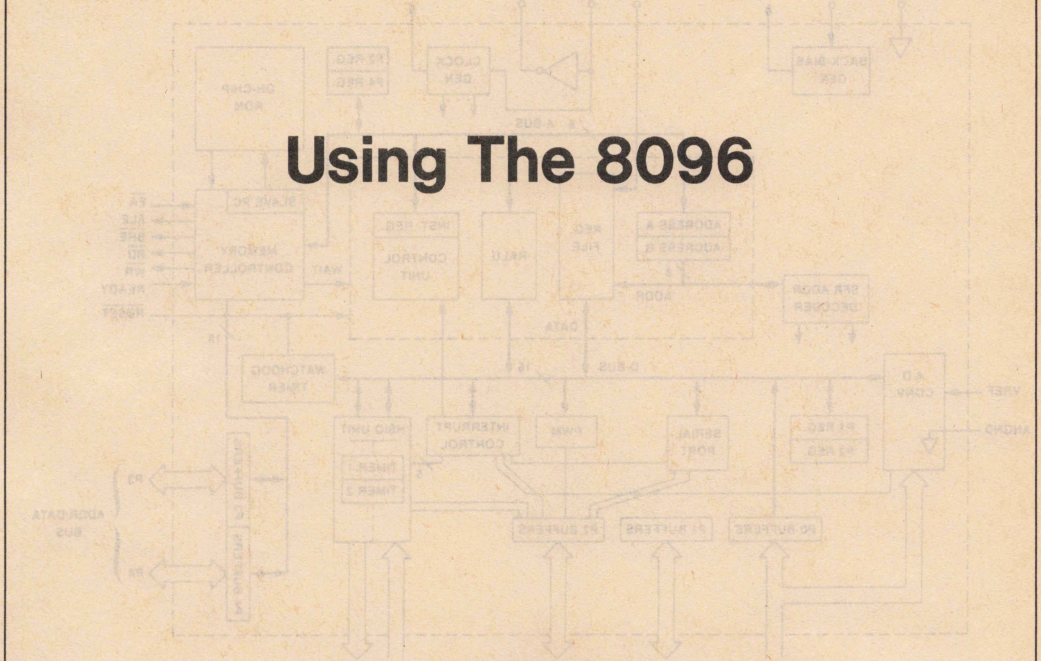
intel

APPLICATION NOTE

AP-248

August 1986

Using The 8096



IRA HORDEN
MCO APPLICATIONS ENGINEER

Order Number: 270061-002

1.0 INTRODUCTION

High speed digital signals are frequently encountered in modern control applications. In addition, there is often a requirement for high speed 16-bit and 32-bit precision in calculations. The MCS®-96 product line, generically referred to as the 8096, is designed to be used in applications which require high speed calculations and fast I/O operations.

The 8096 is a 16-bit microcontroller with dedicated I/O subsystems and a complete set of 16-bit arithmetic instructions including multiply and divide operations. This Ap-note will briefly describe the 8096 in section 2, and then give short examples of how to use each of its key features in section 3. The concluding sections feature a few examples which make use of several chip features simultaneously and some hardware connection suggestions. Further information on the 8096 and its use is available from the sources listed in the bibliography.

2.0 8096 OVERVIEW

2.1. General Description

Unlike microprocessors, microcontrollers are generally optimized for specific applications. Intel's 8048 was optimized for general control tasks while the 8051 was optimized for 8-bit math and single bit boolean operations. The 8096 has been designed for high speed/high performance control applications. Because it has been designed for these applications the 8096 architecture is different from that of the 8048 or 8051.

There are two major sections of the 8096; the CPU section and the I/O section. Each of these sections can be subdivided into functional blocks as shown in Figure 2-1.

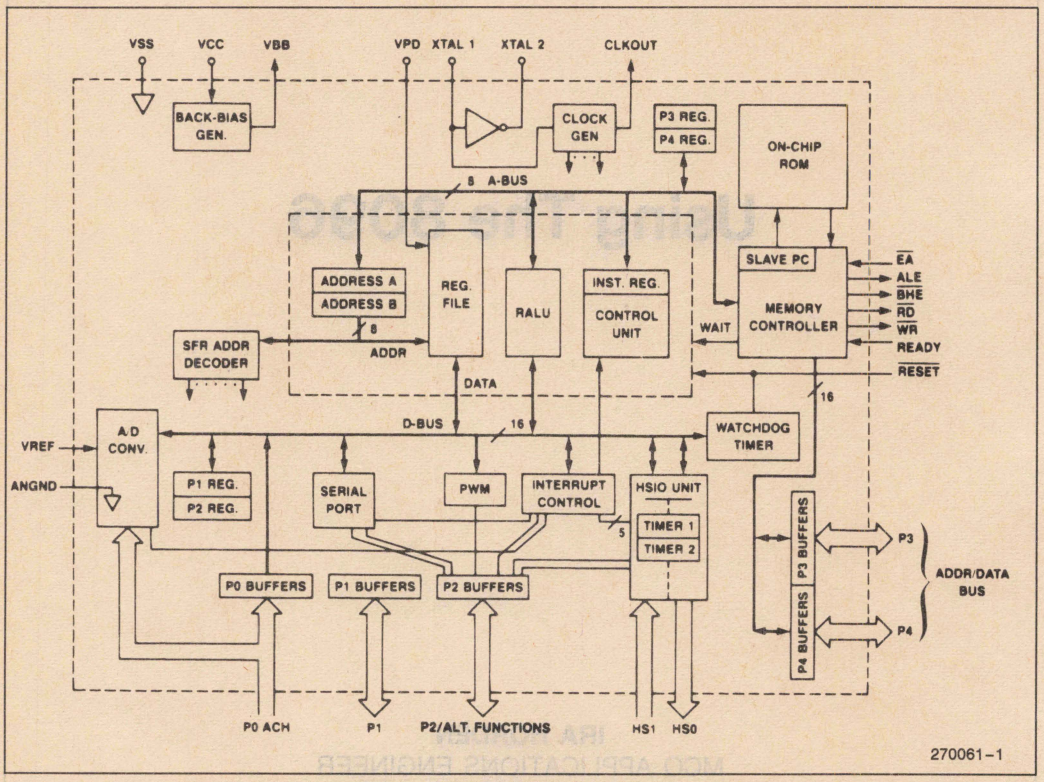


Figure 2-1. 8096 Block Diagram

2.1.1. CPU SECTION

The CPU of the 8096 uses a 16-bit ALU which operates on a 256-byte register file instead of an accumulator. Any of the locations in the register file can be used for sources or destinations for most of the instructions. This is called a register to register architecture. Many of the instructions can also use bytes or words from anywhere in the 64K byte address space as operands. A memory map is shown in Figure 2-2.

In the lower 24 bytes of the register file are the register-mapped I/O control locations, also called Special Function Registers or SFRs. These registers are used to control the on-chip I/O features. The remaining 232 bytes are general purpose RAM, the upper 16 of which can be kept alive using a low current power-down mode.

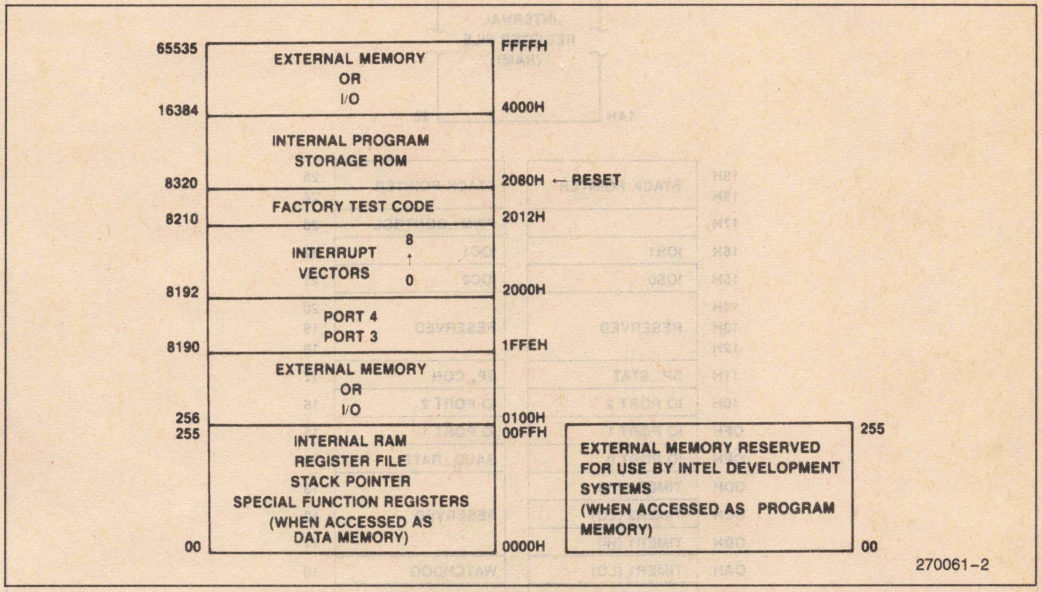


Figure 2-2. Memory Map

Figure 2-3 shows the layout of the register mapped I/O. Some of these registers serve two functions, one if they are read from and another if they are written to. More information about the use of these registers is included in the description of the features which they control.

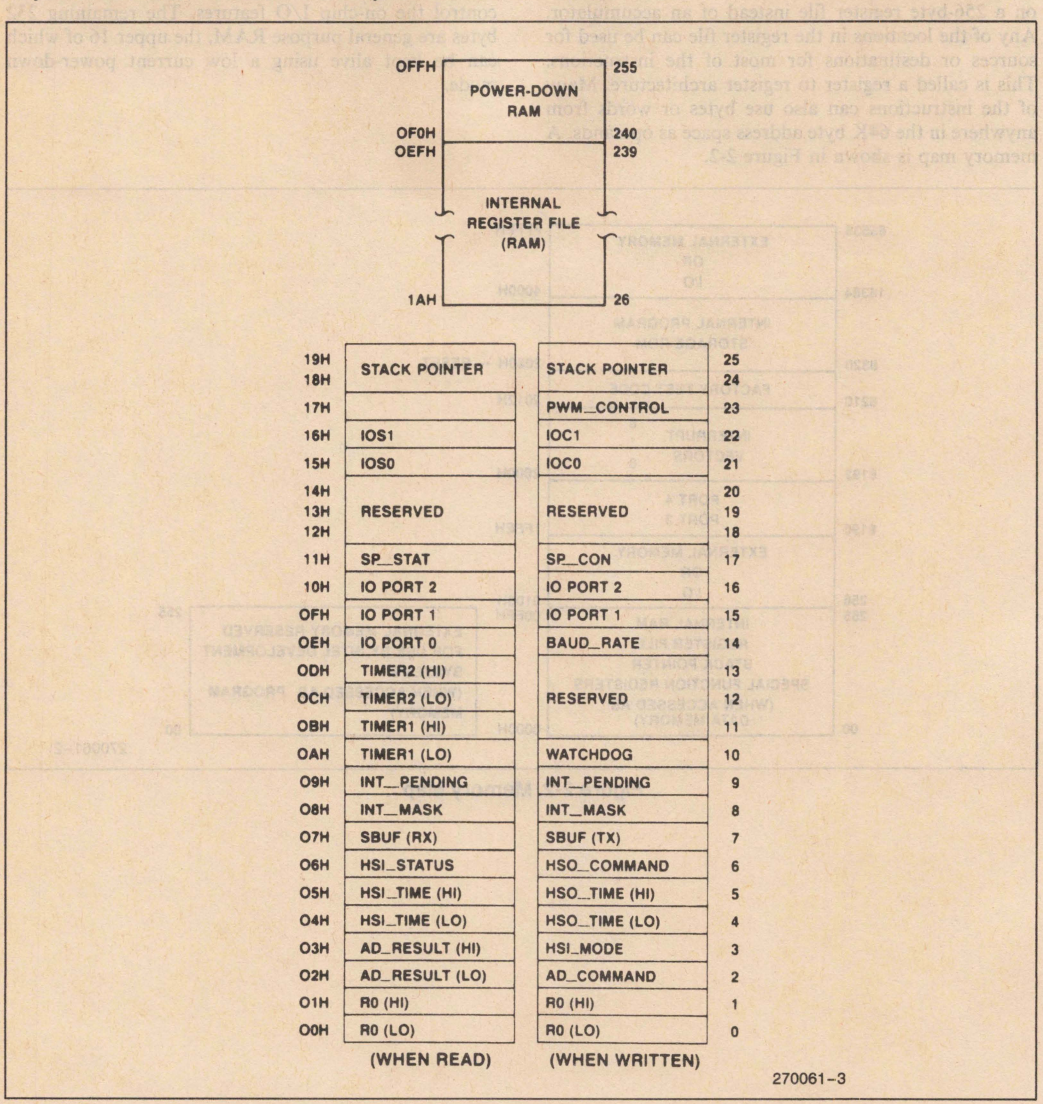


Figure 2-3. SFR Layout

2.1.2. I/O FEATURES

Many of the I/O features on the 8096 are designed to operate with little CPU intervention. A list of the major I/O functions is shown in Figure 2-4. The Watchdog Timer is an internal timer which can be used to reset the system if the software fails to operate properly. The Pulse-Width-Modulation (PWM) output can be used as a rough D to A, a motor driver, or for many other purposes. The A to D converter (ADC) has 8 multiplexed inputs and 10-bit resolution. The serial port has several modes and its own baud rate generator. The High Speed I/O section includes a 16-bit timer, a 16-bit counter, a 4-input programmable edge detector, 4 software timers, and a 6-output programmable event generator. All of these features will be described in section 2.3.

2.2. The Processor Section

2.2.1. OPERATIONS AND ADDRESSING MODES

The 8096 has 100 instructions, some of which operate on bits, some on bytes, some on words and some on longs (double words). All of the standard logical and arithmetic functions are available for both byte and word operations. Bit operations and long operations are provided for some instructions. There are also flag manipulation instructions as well as jump and call instructions. A full set of conditional jumps has been included to speed up testing for various conditions.

Bit operations are provided by the Jump Bit and Jump Not Bit instructions, as well as by immediate masking of bytes. These bit operations can be performed on any of the bytes in the register file or on any of the special function registers. The fast bit manipulation of the SFRs can provide rapid I/O operations.

A symmetric set of byte and word operations make up the majority of the 8096 instruction set. The assembly language for the 8096 (ASM-96) uses a "B" suffix on a mnemonic to indicate a byte operation, without this suffix a word operation is indicated. Many of these operations can have one, two or three operands. An example of a one operand instruction would be:

```
NOT   Value1 ;  Value1 : = 1's complement (Value1)
```

A two operand instruction would have the form:

```
ADD   Value2,Value1 ;  Value2 : = Value2 + Value1
```

A three operand instruction might look like:

```
MUL   Value3,Value2,Value1 ;
                          Value3 : = Value2* Value1
```

The three operand instructions combined with the register to register architecture almost eliminate the necessity of using temporary registers. This results in a faster processing time than machines that have equivalent instruction execution times, but use a standard architecture.

Long (32-bit) operations include shifts, normalize, and multiply and divide. The word divide is a 32-bit by 16-bit operation with a 16-bit quotient and 16-bit remainder. The word multiply is a word by word multiply with a long result. Both of these operations can be done in either the signed or unsigned mode. The direct unsigned modes of these instructions take only 6.5 microseconds. A normalize instruction and sticky bit flag have been included in the instruction set to provide hardware support for the software floating point package (FPAL-96).

Major I/O Functions	
High Speed Input Unit	Provides Automatic Recording of Events
High Speed Output Unit	Provides Automatic Triggering of Events and Real-Time Interrupts
Pulse Width Modulation	Output to Drive Motors or Analog Circuits
A to D Converter	Provides Analog Input
Watchdog Timer	Resets 8096 if a Malfunction Occurs
Serial Port	Provides Synchronous or Asynchronous Link
Standard I/O Lines	Provide Interface to the External World when other Special Features are not needed

Figure 2-4. Major I/O Functions

Mnemonic	Operands	Operation (Note 1)	Flags						Notes
			Z	N	C	V	VT	ST	
ADD/ADDB	2	D ← D + A	↙	↙	↙	↙	↑	—	
ADD/ADDB	3	D ← B + A	↙	↙	↙	↙	↑	—	
ADDC/ADDCB	2	D ← D + A + C	↓	↙	↙	↙	↑	—	
SUB/SUBB	2	D ← D − A	↙	↙	↙	↙	↑	—	
SUB/SUBB	3	D ← B − A	↙	↙	↙	↙	↑	—	
SUBC/SUBCB	2	D ← D − A + C − 1	↓	↙	↙	↙	↑	—	
CMP/CMPB	2	D − A	↙	↙	↙	↙	↑	—	
MUL/MULU	2	D, D + 2 ← D * A	—	—	—	—	—	?	2
MUL/MULU	3	D, D + 2 ← B * A	—	—	—	—	—	?	2
MULB/MULUB	2	D, D + 1 ← D * A	—	—	—	—	—	?	3
MULB/MULUB	3	D, D + 1 ← B * A	—	—	—	—	—	?	3
DIVU	2	D ← (D, D + 2)/A, D + 2 ← remainder	—	—	—	↙	↑	—	2
DIVUB	2	D ← (D, D + 1)/A, D + 1 ← remainder	—	—	—	↙	↑	—	3
DIV	2	D ← (D, D + 2)/A, D + 2 ← remainder	—	—	—	?	↑	—	2
DIVB	2	D ← (D, D + 1)/A, D + 1 ← remainder	—	—	—	?	↑	—	3
AND/ANDB	2	D ← D and A	↙	↙	0	0	—	—	
AND/ANDB	3	D ← B and A	↙	↙	0	0	—	—	
OR/ORB	2	D ← D or A	↙	↙	0	0	—	—	
XOR/XORB	2	D ← D (excl. or) A	↙	↙	0	0	—	—	
LD/LDB	2	D ← A	—	—	—	—	—	—	
ST/STB	2	A ← D	—	—	—	—	—	—	
LDBSE	2	D ← A; D + 1 ← SIGN(A)	—	—	—	—	—	—	3, 4
LDBZE	2	D ← A; D + 1 ← 0	—	—	—	—	—	—	3, 4
PUSH	1	SP ← SP − 2; (SP) ← A	—	—	—	—	—	—	
POP	1	A ← (SP); SP ← SP + 2	—	—	—	—	—	—	
PUSHF	0	SP ← SP − 2; (SP) ← PSW; PSW ← 0000H I ← 0	0	0	0	0	0	0	
POPF	0	PSW ← (SP); SP ← SP + 2; I ← ↙	↙	↙	↙	↙	↙	↙	
SJMP	1	PC ← PC + 11-bit offset	—	—	—	—	—	—	5
LJMP	1	PC ← PC + 16-bit offset	—	—	—	—	—	—	5
BR (indirect)	1	PC ← (A)	—	—	—	—	—	—	
SCALL	1	SP ← SP − 2; (SP) ← PC; PC ← PC + 11-bit offset	—	—	—	—	—	—	5
LCALL	1	SP ← SP − 2; (SP) ← PC; PC ← PC + 16-bit offset	—	—	—	—	—	—	5
RET	0	PC ← (SP); SP ← SP + 2	—	—	—	—	—	—	
J (conditional)	1	PC ← PC + 8-bit offset (if taken)	—	—	—	—	—	—	5
JC	1	Jump if C = 1	—	—	—	—	—	—	5
JNC	1	Jump if C = 0	—	—	—	—	—	—	5
JE	1	Jump if Z = 1	—	—	—	—	—	—	5

Figure 2-5. Instruction Summary

NOTES:
1. If the mnemonic ends in "B", a byte operation is performed, otherwise a word operation is done. Operands D, B, and A must conform to the alignment rules for the required operand type. D and B are locations in the register file; A can be located anywhere in memory.
2. D, D + 2 are consecutive WORDS in memory; D is DOUBLE-WORD aligned.
3. D, D + 1 are consecutive BYTES in memory; D is WORD aligned.
4. Changes a byte to a word.
5. Offset is a 2's complement number.

Mnemonic	Oper-ands	Operation (Note 1)	Flags						Notes
			Z	N	C	V	VT	ST	
JNE	1	Jump if Z = 0	—	—	—	—	—	—	5
JGE	1	Jump if N = 0	—	—	—	—	—	—	5
JLT	1	Jump if N = 1	—	—	—	—	—	—	5
JGT	1	Jump if N = 0 and Z = 0	—	—	—	—	—	—	5
JLE	1	Jump if N = 1 or Z = 1	—	—	—	—	—	—	5
JH	1	Jump if C = 1 and Z = 0	—	—	—	—	—	—	5
JNH	1	Jump if C = 0 or Z = 1	—	—	—	—	—	—	5
JV	1	Jump if V = 1	—	—	—	—	—	—	5
JNV	1	Jump if V = 0	—	—	—	—	—	—	5
JVT	1	Jump if VT = 1; Clear VT	—	—	—	—	0	—	5
JNVT	1	Jump if VT = 0; Clear VT	—	—	—	—	0	—	5
JST	1	Jump if ST = 1	—	—	—	—	—	—	5
JNST	1	Jump if ST = 0	—	—	—	—	—	—	5
JBS	3	Jump if Specified Bit = 1	—	—	—	—	—	—	5, 6
JBC	3	Jump if Specified Bit = 0	—	—	—	—	—	—	5, 6
DJNZ	1	D ← D − 1; if D ≠ 0 then PC ← PC + 8-bit offset	—	—	—	—	—	—	5
DEC/DECB	1	D ← D − 1	↙	↙	↙	↙	↑	—	
NEG/NEGB	1	D ← 0 − D	↙	↙	↙	↙	↑	—	
INC/INCB	1	D ← D + 1	↙	↙	↙	↙	↑	—	
EXT	1	D ← D; D + 2 ← Sign (D)	↙	↙	0	0	—	—	2
EXTB	1	D ← D; D + 1 ← Sign (D)	↙	↙	0	0	—	—	3
NOT/NOTB	1	D ← Logical Not (D)	↙	↙	0	0	—	—	
CLR/CLRB	1	D ← 0	1	0	0	0	—	—	
SHL/SHLB/SHLL	2	C ← msb — — — — — lsb ← 0	↙	?	↙	↙	↑	—	7
SHR/SHRB/SHRL	2	0 → msb — — — — — lsb → C	↙	?	↙	0	—	↙	7
SHRA/SHRAB/SHRAL	2	msb → msb — — — — — lsb → C	↙	↙	↙	0	—	↙	7
SETC	0	C ← 1	—	—	1	—	—	—	
CLRC	0	C ← 0	—	—	0	—	—	—	
CLRVT	0	VT ← 0	—	—	—	—	0	—	
RST	0	PC ← 2080H	0	0	0	0	0	0	8
DI	0	Disable All Interrupts (I ← 0)	—	—	—	—	—	—	
EI	0	Enable All Interrupts (I ← 1)	—	—	—	—	—	—	
NOP	0	PC ← PC + 1	—	—	—	—	—	—	
SKIP	0	PC ← PC + 2	—	—	—	—	—	—	
NORML	2	Left Shift Till msb = 1; D ← shift count	↙	?	0	—	—	—	7
TRAP	0	SP ← SP − 2; (SP) ← PC PC ← (2010H)	—	—	—	—	—	—	9

Figure 2-5. Instruction Summary (Continued)

NOTES:
1. If the mnemonic ends in "B", a byte operation is performed, otherwise a word operation is done. Operands D, B, and A must conform to the alignment rules for the required operand type. D and B are locations in the register file; A can be located anywhere in memory.
5. Offset is a 2's complement number.
6. Specified bit is one of the 2048 bits in the register file.
7. The "L" (Long) suffix indicates double-word operation.
8. Initiates a Reset by pulling RESET low. Software should re-initialize all the necessary registers with code starting at 2080H.
9. The assembler will not accept this mnemonic.

One operand of most of the instructions can be used with any one of six addressing modes. These modes increase the flexibility and overall execution speed of the 8096. The addressing modes are: register-direct, immediate, indirect, indirect with auto-increment, and long and short indexed.

The fastest instruction execution is gained by using either register direct or immediate addressing. Register-direct addressing is similar to normal direct addressing, except that only addresses in the register file or SFRs can be addressed. The indexed mode is used to directly address the remainder of the 64K address space. Immediate addressing operates as would be expected, using the data following the opcode as the operand.

Both of the indirect addressing modes use the value in a word register as the address of the operand. If the indirect auto-increment mode is used then the word register is incremented by one after a byte access or by two after a word access. This mode is particularly useful for accessing lookup tables.

Access to any of the locations in the 64K address space can be obtained by using the long indexed addressing mode. In this mode a 16-bit 2's complement value is added to the contents of a word register to form the address of the operand. By using the zero register as the index, ASM96 (the assembler) can accept "direct" addressing to any location. The zero register is located at 0000H and always has a value of zero. A short indexed mode is also available to save some time and code. This mode uses an 8-bit 2's complement number as the offset instead of a 16-bit number.

2.2.2. ASSEMBLY LANGUAGE

The multiple addressing modes of the 8096 make it easy to program in assembly language and provide an excellent interface to high level languages. The instructions accepted by the assembler consist of mnemonics followed by either addresses or data. A list of the mnemonics and their functions are shown in Figure 2-5. The addresses or data are given in different formats depending on the addressing mode. These modes and formats are shown in Figure 2-6.

Additional information on 8096 assembly language is available in the MCS-96 Macro Assembler Users Guide, listed in the bibliography.

```
Mnem  Dest or Src1              ; One operand direct
Mnem  Dest, Src1                ; Two operand direct
Mnem  Dest, Src1, Src2          ; Three operand direct

Mnem  #Src1                     ; One operand immediate
Mnem  Dest, #Src1               ; Two operand immediate
Mnem  Dest, Src1, #Src2         ; Three operand immediate

Mnem  [addr]                    ; One operand indirect
Mnem  [addr]+                   ; One operand indirect auto-increment
Mnem  Dest, [addr]              ; Two operand indirect
Mnem  Dest, [addr]+             ; Two operand indirect auto-increment
Mnem  Dest, Src1, [addr]        ; Three operand indirect
Mnem  Dest, Src1, [addr]+       ; Three operand indirect auto-increment

Mnem  Dest, offs [addr]         ; Two operand indexed (short or long)
Mnem  Dest, Src1, offs [addr]   ; Three operand indexed (short or long)

Where: "Mnem" is the instruction mnemonic
       "Dest" is the destination register
       "Src1", "Src2" are the source registers
       "addr" is a register containing a value to be used in computing the address of an operand
       "offs" is an offset used in computing the address of an operand
```

270061-B3

Figure 2-6. Instruction Format

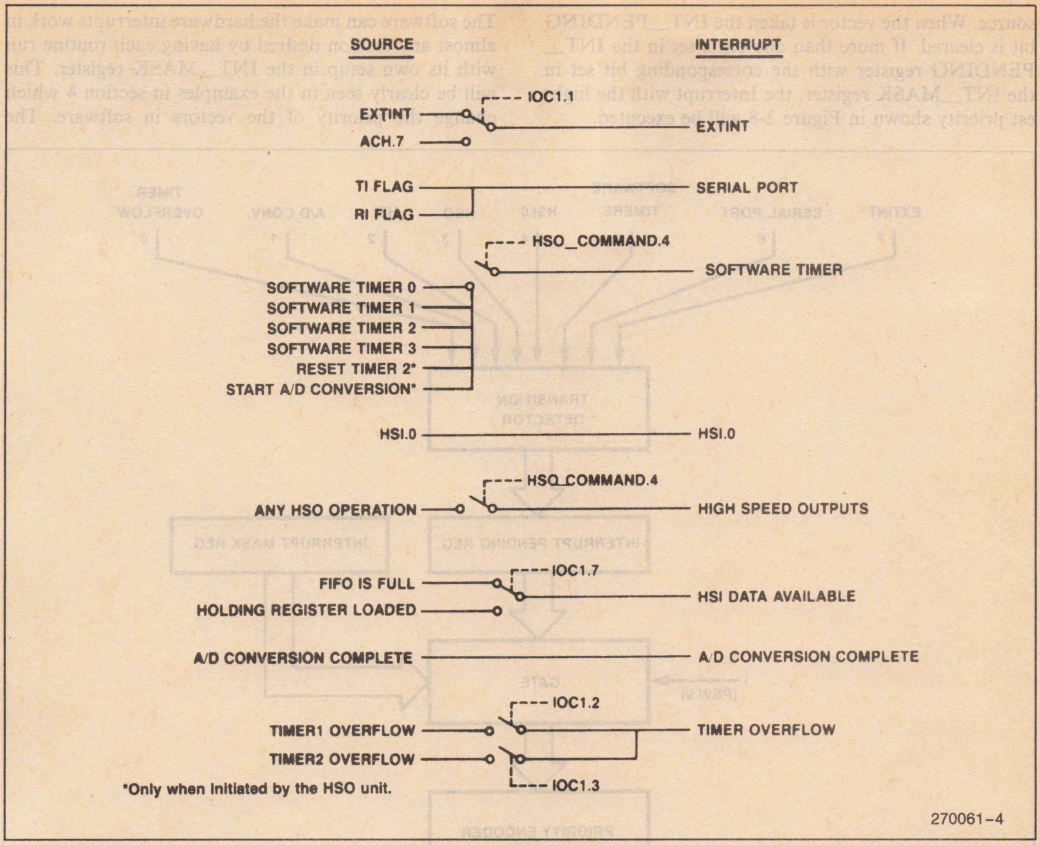


Figure 2-7. Interrupt Sources

2.2.3. INTERRUPTS

The flexibility of the instruction set is carried through into the interrupt system. There are 20 different interrupt sources that can be used on the 8096. The 20 sources vector through 8 locations or interrupt vectors. The vector names and their sources are shown in Figure 2-7, with their locations listed in Figure 2-8. Control of the interrupts is handled through the Interrupt Pending Register (INT_PENDING), the Interrupt Mask Register (INT_MASK), and the I bit in the PSW (PSW.9). Figure 2-9 shows a block diagram of the interrupt structure. The INT_PENDING register contains bits which get set by hardware when an interrupt occurs. If the interrupt mask register bit for that source is a 1 and PSW.9 = 1, a vector will be taken to the address listed in the interrupt vector table for that

Source	Vector Location		Priority
	(High Byte)	(Low Byte)	
Software	2011H	2010H	Not Applicable
Extint	200FH	200EH	7 (Highest)
Serial Port	200DH	200CH	6
Software Timers	200BH	200AH	5
HSI.0	2009H	2008H	4
High Speed Outputs	2007H	2006H	3
HSI Data Available	2005H	2004H	2
A/D Conversion Complete	2003H	2002H	1
Timer Overflow	2001H	2000H	0 (Lowest)

Figure 2-8. Interrupt Vectors and Priorities

source. When the vector is taken the INT_PENDING bit is cleared. If more than one bit is set in the INT_PENDING register with the corresponding bit set in the INT_MASK register, the Interrupt with the highest priority shown in Figure 2-8 will be executed.

The software can make the hardware interrupts work in almost any fashion desired by having each routine run with its own setup in the INT_MASK register. This will be clearly seen in the examples in section 4 which change the priority of the vectors in software. The

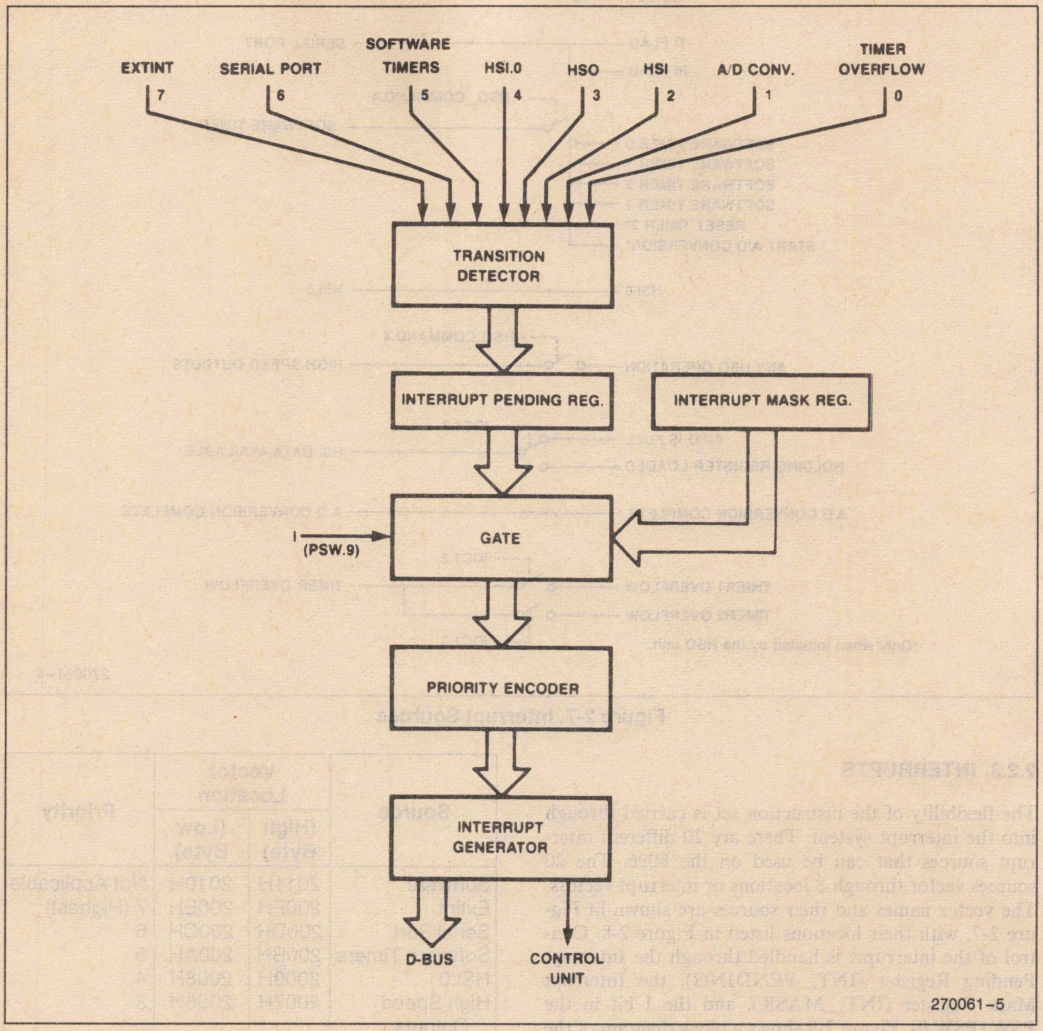


Figure 2-9. Interrupt Structure Block Diagram

15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
Z	N	V	VT	C	—	I	ST	INT_MASK							

WHERE:

Z is the zero flag. It is set when the result of an operation is zero.

N is the negative flag. It is set to the algebraically correct sign of the result regardless of overflows.

V is the overflow flag. It is set if an overflow occurs.

VT is the overflow trap flag. It is set when the VT flag is set and cleared by JVT, JNVT, or CLRVT.

C is the carry flag. It is set if a carry was generated by the prior operation.

I is the global interrupt enable bit.

ST is the sticky bit. It is set during a right shift if a one was shifted into and then out of the carry flag.

INT_MASK is the interrupt mask register and contains bits which individually enable the 8 interrupt vectors.

Figure 2-10. The PSW Register

PSW (shown in Figure 2-10), stores the INT_MASK register in its lower byte so that the mask register can be pushed and popped along with the machine status when moving in and out of routines. The action of pushing flags clears the PSW which includes PSW.9, the interrupt enable bit. Therefore, after a PUSHF instruction interrupts are disabled. In most cases an interrupt service routine will have the basic structure shown below.

```
INT     VECTOR:

  PUSHF
  LDB  INT_MASK, #xxxxxxxxB
  EI
   -
   -  ;Insert service routine here
   -
  POPF
  RET
```

The PUSHF instruction saves the PSW including the old INT_MASK register. The PSW, including the interrupt enable bit are left cleared. If some interrupts need to be enabled while the service routine runs, the INT_MASK is loaded with a new value and interrupts are globally enabled before the service routine continues. At the end of the service routine a POPF instruction is executed to restore the old PSW. The RET instruction is executed and the code returns to the desired location. Although the POPF instruction can enable the interrupts the next instruction will always execute. This prevents unnecessary building of the stack by ensuring that the RET always executes before another interrupt vector is taken.

2.3. On-Chip I/O Section

All of the on-chip I/O features of the 8096 can be accessed through the special function registers, as shown in Figure 2-3. The advantage of using register-mapped I/O is that these registers can be used as the sources or destinations of CPU operations. There are seven major I/O functions. Each one of these will be considered with a section of code to exemplify its usage. The first section covered will be the High Speed I/O, (HSIO), subsystem. This section includes the High Speed Input (HSI) unit, High Speed Output (HSO) unit, and the Timer/Counter section.

2.3.1. TIMER/COUNTERS

The 8096 has two time bases, Timer 1 and Timer 2. Timer 1 is a 16-bit free running timer which is incremented every 8 state times. (A state time is 3 oscillator periods, or 0.25 microseconds with a 12 MHz crystal.)

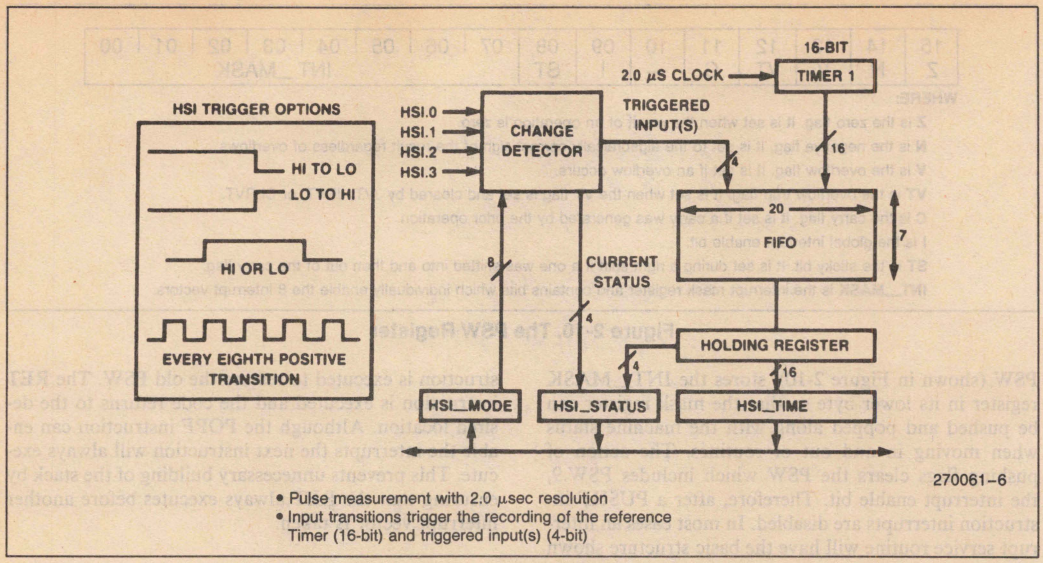


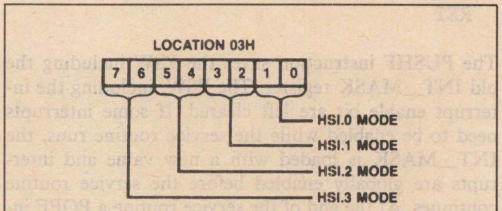
Figure 2-11. HSI Unit Block Diagram

Its value can be read at any time and used as a reference for both the HSI section and the HSO section. Timer 1 can cause an interrupt when it overflows, and cannot be modified or stopped without resetting the entire chip. Timer 2 is really an event counter since it uses an external clock source. Like Timer 1, it is 16-bits wide, can be read at any time, can be used with the HSO section, and can generate an interrupt when it overflows. Control of Timer 2 is limited to incrementing it and resetting it. Specific values can not be written to it.

Although the 8096 has only two timers, the timer flexibility is equal to a unit with many timers thanks to the HSIO unit. The HSI enables one to measure times of external events on up to four lines using Timer 1 as a timer base. The HSO unit can schedule and execute internal events and up to six external events based on the values in either Timer 1 or Timer 2. The 8096 also includes separate, dedicated timers for the baud rate generator and watchdog timer.

2.3.2. HSI

The HSI unit can be thought of as a message taker which records the line which had an event and the time at which the event occurred. Four types of events can trigger the HSI unit, as shown in the HSI block diagram in Figure 2-11. The HSI unit can measure pulse widths and record times of events with a 2



Where each 2-bit mode control field defines one of 4 possible modes:

00	8 positive transitions
01	Each positive transition
10	Each negative transition
11	Every transition (positive and negative)

Figure 2-12. HSI Mode Register

microsecond resolution. It can look for one of four events on each of four lines simultaneously, based on the information in the HSI Mode register, shown in Figure 2-12. The information is then stored in a seven level FIFO for later retrieval. Whenever the FIFO contains information, the earliest entry is placed in the holding register. When the holding register is read, the next valid piece of information is loaded into it. Interrupts can be generated by the HSI unit at the time the holding register is loaded or when the FIFO has six or more entries.

2.3.3. HSO

Just as the HSI can be thought of as a message taker, the HSO can be thought of as a message sender. At times determined by the software, the HSO sends mes-

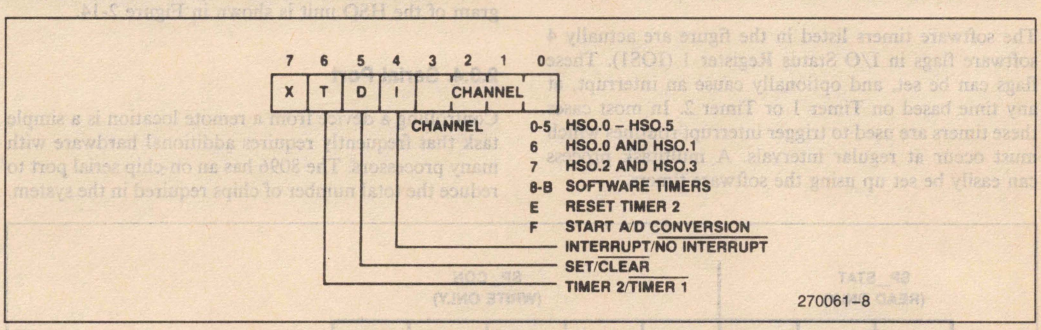


Figure 2-13. HSO Command Register

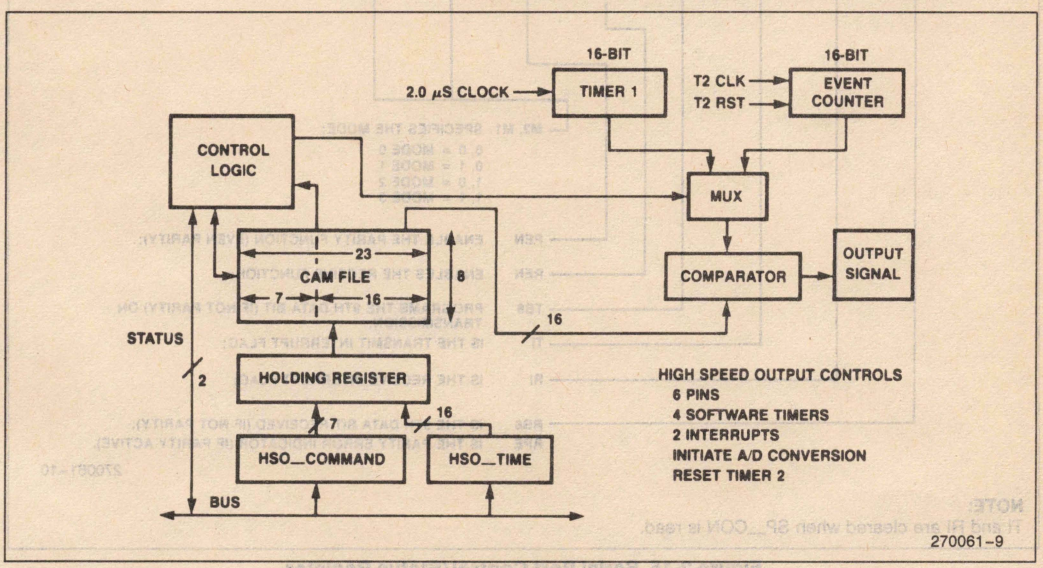


Figure 2-14. HSO Block Diagram

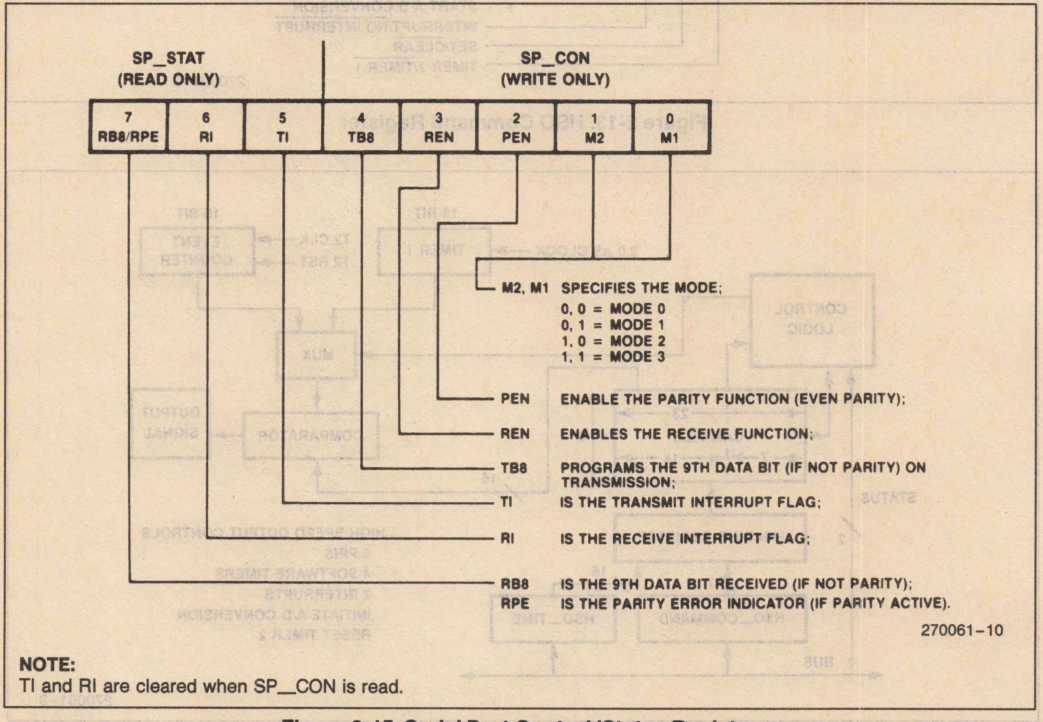
sages to various devices to have them turn on, turn off, start processing, or reset. Since the programmed times can be referenced to either Timer 1 or Timer 2, the HSO makes the two timers look like many. For example, if several events have to occur at specific times, the HSO unit can schedule all of the events based on a single timer. The events that can be scheduled to occur and the format of the command written to the HSO Command register are shown in Figure 2-13.

The software timers listed in the figure are actually 4 software flags in I/O Status Register 1 (IOS1). These flags can be set, and optionally cause an interrupt, at any time based on Timer 1 or Timer 2. In most cases these timers are used to trigger interrupt routines which must occur at regular intervals. A multitask process can easily be set up using the software timers.

A CAM (Content Addressable Memory) file is the main component of the HSO. This file stores up to eight events which are pending to occur. Every state time one location of the CAM is compared with the two timers. After 8 state times, (two microseconds with a 12 MHz clock), the entire CAM has been searched for time matches. If a match occurs the specified event will be triggered and that location of the CAM will be made available for another pending event. A block diagram of the HSO unit is shown in Figure 2-14.

2.3.4. Serial Port

Controlling a device from a remote location is a simple task that frequently requires additional hardware with many processors. The 8096 has an on-chip serial port to reduce the total number of chips required in the system.



NOTE:
TI and RI are cleared when SP_CON is read.

Figure 2-15. Serial Port Control/Status Register

The serial port is similar to that on the MCS-51 product line. It has one synchronous and three asynchronous modes. In the asynchronous modes baud rates of up to 187.5 Kbaud can be used, while in the synchronous mode rates up to 1.5 Mbaud are available. The chip has a baud rate generator which is independent of Timer 1 and Timer 2, so using the serial port does not take away any of the HSI, HSO or timer flexibility or functionality.

Control of the serial port is provided through the SPCON/SPSTAT (Serial Port CONtrol/Serial Port STATus) register. This register, shown in Figure 2-15, has some bits which are read only and others which are write only. Although the functionality of the port is similar to that of the 8051, the names of some of the modes and control bits are different. The way in which the port is used from a software standpoint is also slightly different since RI and TI are cleared after each read of the register.

The four modes of the serial port are referred to as modes 0, 1, 2 and 3. Mode 0 is the synchronous mode, and is commonly used to interface to shift registers for I/O expansion. In this mode the port outputs a pulse train on the TXD pin and either transmits or receives data on the RXD pin. Mode 1 is the standard asynchronous mode, 8 bits plus a stop and start bit are sent or received. Modes 2 and 3 handle 9 bits plus a stop and start bit. The difference between the two is, that in Mode 2 the serial port interrupt will not be activated unless the ninth data bit is a one; in Mode 3 the interrupt is activated whenever a byte is received. These two modes are commonly used for interprocessor communication.

Using XTAL1:

$$\text{Mode 0: } \frac{\text{Baud}}{\text{Rate}} = \frac{\text{XTAL1 frequency}}{4*(B+1)};\ B \neq 0$$

$$\text{Others: } \frac{\text{Baud}}{\text{Rate}} = \frac{\text{XTAL1 frequency}}{64*(B+1)}$$

Using T2CLK:

$$\text{Mode 0: } \frac{\text{Baud}}{\text{Rate}} = \frac{\text{T2CLK frequency}}{B};\ B \neq 0$$

$$\text{Others: } \frac{\text{Baud}}{\text{Rate}} = \frac{\text{T2CLK frequency}}{16*B};\ B \neq 0$$

Note that B cannot equal 0, except when using XTAL1 in other than mode 0.

Figure 2-16. Baud Rate Formulas

Baud rates for all of the modes are controlled through the Baud Rate register. This is a byte wide register which is loaded sequentially with two bytes, and internally stores the value as a word. The least significant byte is loaded to the register followed by the most significant. The most significant bit of the baud value determines the clock source for the baud rate generator. If the bit is a one, the XTAL1 pin is used as the source, if it is a zero, the T2 CLK pin is used. The formulas shown in Figure 2-16 can be used to calculate the baud rates. The variable "B" is used to represent the least significant 15 bits of the value loaded into the baud rate register.

The baud rate register values for common baud rates are shown in Figure 2-17. These values can be used when XTAL1 is selected as the clock source for serial modes other than Mode 0. The percentage deviation from theoretical is listed to help assess the reliability of a given setup. In most cases a serial link will work if there is less than a 2.5% difference between the baud rates of the two systems. This is based on the assumption that 10 bits are transmitted per frame and the last bit of the frame must be valid for at least six-eights of the bit time. If the two systems deviate from theoretical by 1.25% in opposite directions the maximum tolerance of 2.5% will be reached. Therefore, caution must be used when the baud rate deviation approaches 1.25% from theoretical. Note that an XTAL1 frequency of 11.0592 MHz can be used with the table values for 11 MHz to provide baud rates that have 0.0 percent deviation from theoretical. In most applications, however, the accuracy available when using an 11 MHz input frequency is sufficient.

Serial port Mode 1 is the easiest mode to use as there is little to worry about except initialization and loading and unloading SBUF, the Serial port BUFfer. If parity is enabled, (i.e., PEN = 1), 7 bits plus even parity are used instead of 8 data bits. The parity calculation is done in hardware for even parity. Modes 2 and 3 are similar to Mode 1, except that the ninth bit needs to be controlled and read. It is also not possible to enable parity in Mode 2. When parity is enabled in Mode 3 the ninth bit becomes the parity bit. If parity is not enabled, (i.e., PEN = 0), the TB8 bit controls the state of the ninth transmitted bit. This bit must be set prior to each transmission. On reception, if PEN = 0, the RB8 bit indicates the state of the ninth received bit. If parity is enabled, (i.e., PEN = 1), the same bit is called RPE (Receive Parity Error), and is used to indicate a parity error.

XTAL1 Frequency = 12.0 MHz		
Baud Rate	**Baud Register Value**	**Percent Error**
19.2K	8009H	+2.40
9600	8013H	+2.40
4800	8026H	−0.16
2400	804DH	−0.16
1200	809BH	−0.16
300	8270H	0.00
XTAL1 Frequency = 11.0 MHz		
19.2K	8008H	+0.54
9600	8011H	+0.54
4800	8023H	+0.54
2400	8047H	+0.54
1200	808EH	−0.16
300	823CH	+0.01
XTAL1 Frequency = 10.0 MHz		
19.2K	8007H	−1.70
9600	800FH	−1.70
4800	8020H	+1.38
2400	8040H	−0.16
1200	8081H	−0.16
300	8208H	+0.03

Figure 2-17. Baud Rate Values for 10, 11, 12 MHz

The software used to communicate between processors is simplified by making use of Modes 2 and 3. In a basic protocol the ninth bit is called the address bit. If it is set high then the information in that byte is either the address of one of the processors on the link, or a command for all the processors. If the bit is a zero, the byte contains information for the processor or processors previously addressed. In standby mode all processors wait in Mode 2 for a byte with the address bit set. When they receive that byte, the software determines if the next message is for them. The processor that is to receive the message switches to Mode 3 and receives the information. Since this information is sent with the ninth bit set to zero, none of the processors set to Mode 2 will be interrupted. By using this scheme the overall CPU time required for the serial port is minimized.

A typical connection diagram for the multi-processor mode is shown in Figure 2-18. This type of communicaton can be used to connect peripherals to a desk top computer, the axis of a multi-axis machine, or any other group of microcontrollers jointly performing a task.

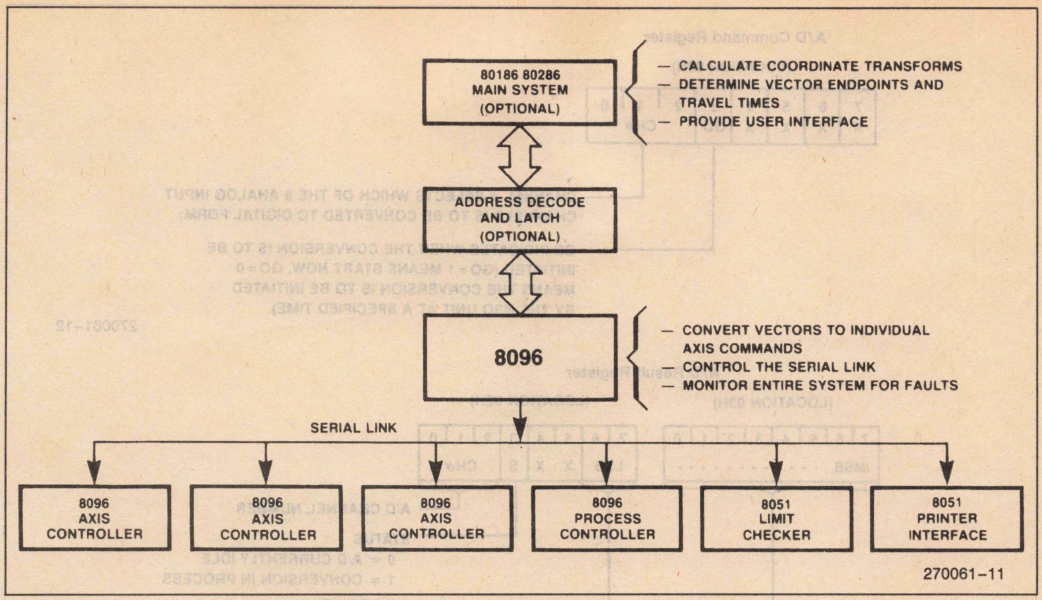


Figure 2-18. Multiprocessor Communication

Mode 0, the synchronous mode, is typically used for interfacing to shift registers for I/O expansion. The software to control this mode involves the REN (Receiver ENable) bit, the clearing of the RI bit, and writing to SBUF. To transmit to a shift register, REN is set to zero and SBUF is loaded with the information. The information will be sent and then the TI flag will be set. There are two ways to cause a reception to begin. The first is by causing a rising edge to occur on the REN bit, the second is by clearing RI with REN = 1. In either case, RI is set again when the received byte is available in SBUF.

2.3.5. A to D CONVERTER

Analog inputs are frequently required in a microcontroller application. The 8097 has a 10-bit A to D converter that can use any one of eight input channels. The conversions are done using the successive approximation method, and require 168 state times (42 microseconds with a 12 MHz clock.)

The results are guaranteed monotonic by design of the converter. This means that if the analog input voltage changes, even slightly, the digital value will either stay the same or change in the same direction as the analog input. When doing process control algorithms, it is frequently the changes in inputs that are required, not the absolute accuracy of the value. For this reason, even if the absolute accuracy of a 10-bit converter is the same as that of an 8-bit converter, the 10-bit monotonic converter is much more useful.

Since most of the analog inputs which are monitored by a microcontroller change very slowly relative to the 42 microsecond conversion time, it is acceptable to use a capacitive filter on each input instead of a sample and hold. The 8097 does not have an internal sample and hold, so it is necessary to ensure that the input signal does not change during the conversion time. The input to the A/D must be between ANGND and VREF. ANGND must be within a few millivolts of VSS and VREF must be within a few tenths of a volt of VCC.

Using the A to D converter on the 8097 can be a very low software overhead task because of the interrupt and HSO unit structure. The A to D can be started by the HSO unit at a preset time. When the conversion is complete it is possible to generate an interrupt. By using these features the A to D can be run under complete interrupt control. The A to D can also be directly

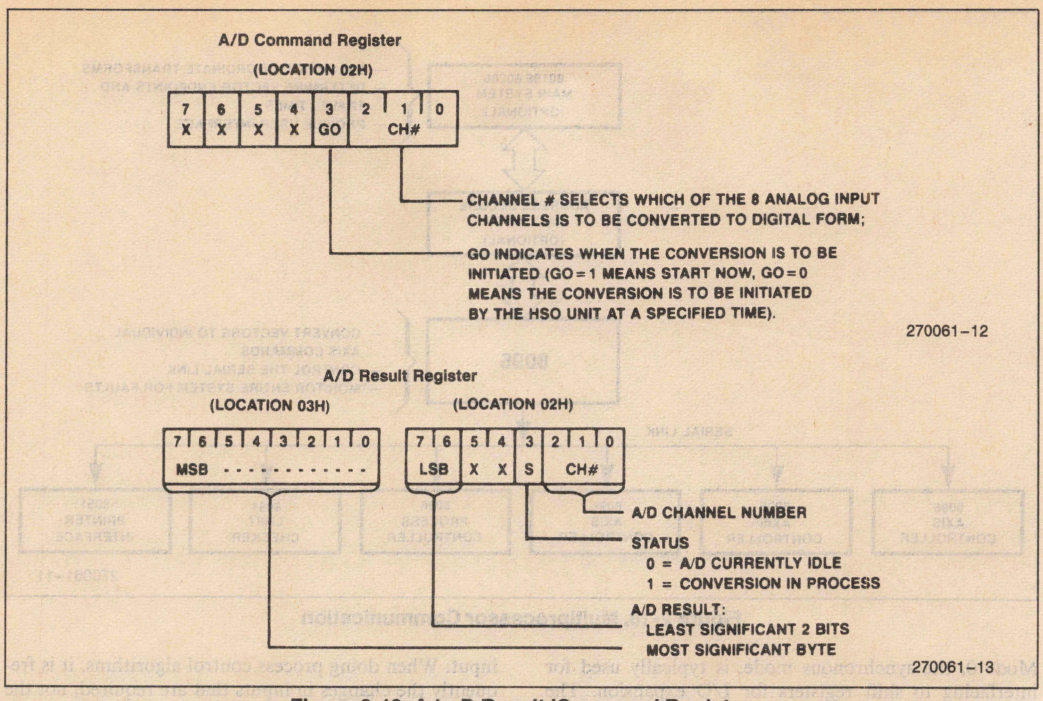


Figure 2-19. A to D Result/Command Register

controlled by software flags which are located in the AD_RESULT/AD_COMMAND Register, shown in Figure 2-19.

2.3.6. PWM REGISTER

Analog outputs are just as important as analog inputs when connecting to a piece of equipment. True digital to analog converters are difficult to make on a microprocessor because of all of the digital noise and the necessity of providing an on chip, relatively high current, rail to rail driver. They also take up a fair amount of silicon area which can be better used for other features. The A to D converter does use a D to A, but the currents involved are very small.

For many applications an analog output signal can be replaced by a Pulse Width Modulated (PWM) signal. This signal can be easily generated in hardware, and takes up much less silicon area than a true D to A. The signal is a variable duty cycle, fixed frequency waveform that can be integrated to provide an approximation to an analog output. The frequency is fixed at a period of 64 microseconds for a 12 MHz clock speed. Controlling the PWM simply requires writing the desired duty cycle value (an 8-bit value) to the PWM Register. Some typical output waveforms that can be generated are shown in Figure 2-20.

Converting the PWM signal to an analog signal varies in difficulty, depending upon the requirements of the system. Some systems, such as motors or switching power supplies actually require a PWM signal, not a true analog one. For many other cases it is necessary only to amplify the signal so that it switches rail-to-rail, and then filter it. Switching rail-to-rail means that the output of the amplifier will be a reference value when the input is a logical one, and the output will

be zero when the input is a logical zero. The filter can be a simple RC network or an active filter. If a large amount of current is needed a buffer is also required. For low output currents, (less than 100 microamps or so), the circuit shown in Figure 2-21 can be used.

The RC network determines how quiet the output is, but the quieter the output, the slower it can change. The design of high accuracy voltage followers and active filters is beyond the scope of this paper, however many books on the subject are available.

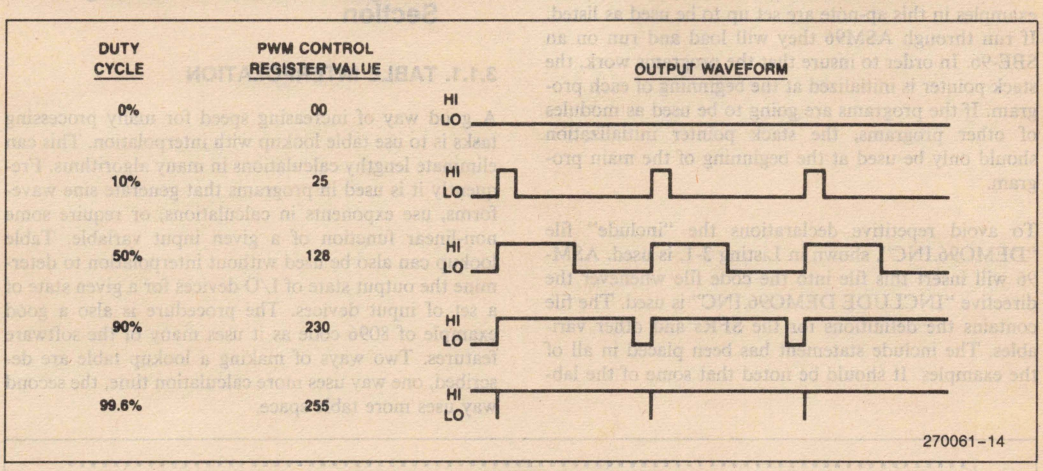


Figure 2-20. PWM Output Waveforms

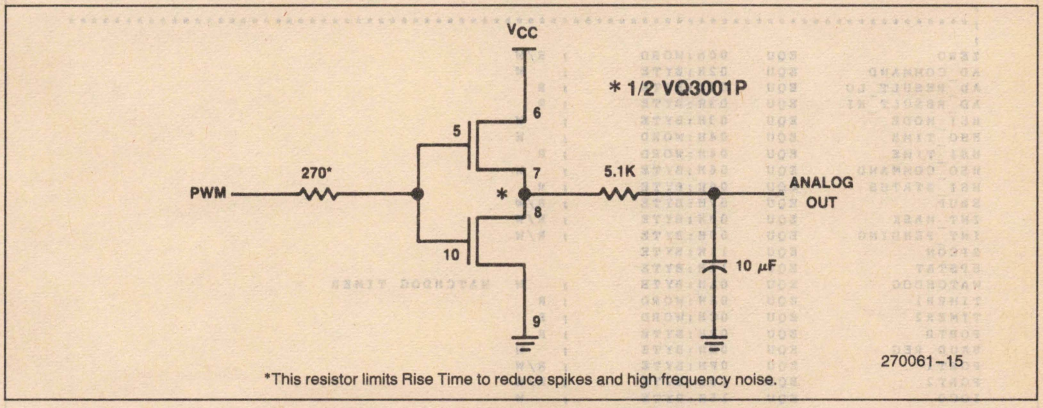


Figure 2-21. PWM to Analog Conversion Circuitry

3.0 BASIC SOFTWARE EXAMPLES

The examples in this section show how to use each I/O feature individually. Examples of using more than one feature at a time are described in section 4. All of the examples in this ap-note are set up to be used as listed. If run through ASM96 they will load and run on an SBE-96. In order to insure that the programs work, the stack pointer is initialized at the beginning of each program. If the programs are going to be used as modules of other programs, the stack pointer initialization should only be used at the beginning of the main program.

To avoid repetitive declarations the "include" file "DEMO96.INC", shown in Listing 3-1, is used. ASM-96 will insert this file into the code file whenever the directive "INCLUDE DEMO96.INC" is used. The file contains the definitions for the SFRs and other variables. The include statement has been placed in all of the examples. It should be noted that some of the labels in this file are different from those in the file 8096.INC that is provided in the ASM-96 package.

3.1. Using the 8096's Processing Section

3.1.1. TABLE INTERPOLATION

A good way of increasing speed for many processing tasks is to use table lookup with interpolation. This can eliminate lengthy calculations in many algorithms. Frequently it is used in programs that generate sine waveforms, use exponents in calculations, or require some non-linear function of a given input variable. Table lookup can also be used without interpolation to determine the output state of I/O devices for a given state of a set of input devices. The procedure is also a good example of 8096 code as it uses many of the software features. Two ways of making a lookup table are described, one way uses more calculation time, the second way uses more table space.

```
;*******************************************************************************
;
; DEMO96.INC - DEFINITION OF SYMBOLIC NAMES FOR THE I/O REGISTERS OF THE 8096
;
;*******************************************************************************
;
ZERO            EQU   00h:WORD      ; R/W
AD_COMMAND      EQU   02H:BYTE      ;   W
AD_RESULT_LO    EQU   02H:BYTE      ; R
AD_RESULT_HI    EQU   03H:BYTE      ; R
HSI_MODE        EQU   03H:BYTE      ;   W
HSO_TIME        EQU   04H:WORD      ;   W
HSI_TIME        EQU   04H:WORD      ; R
HSO_COMMAND     EQU   06H:BYTE      ;   W
HSI_STATUS      EQU   06H:BYTE      ; R
SBUF            EQU   07H:BYTE      ; R/W
INT_MASK        EQU   08H:BYTE      ; R/W
INT_PENDING     EQU   09H:BYTE      ; R/W
SPCON           EQU   11H:BYTE
SPSTAT          EQU   11H:BYTE
WATCHDOG        EQU   0AH:BYTE      ;   W  WATCHDOG TIMER
TIMER1          EQU   0AH:WORD      ; R
TIMER2          EQU   0CH:WORD      ; R
PORT0           EQU   0EH:BYTE      ; R
BAUD_REG        EQU   0EH:BYTE      ;   W
PORT1           EQU   0FH:BYTE      ; R/W
PORT2           EQU   10H:BYTE      ; R/W
IOC0            EQU   15H:BYTE      ;   W
IOS0            EQU   15H:BYTE      ; R
IOC1            EQU   16H:BYTE      ;   W
IOS1            EQU   16H:BYTE      ; R
PWM_CONTROL     EQU   17H:BYTE      ;   W
SP              EQU   18H:WORD      ; R/W STACK POINTER


RSEG at 1CH

        AX:     DSW     1
        DX:     DSW     1
        BX:     DSW     1
        CX:     DSW     1

        AL      EQU     AX      :BYTE
        AH      EQU     (AX+1)  :BYTE
```

270061-16

Listing 3-1. Include File DEMO.96.INC

In both methods the procedure is similar. Values of a function are stored in memory for specific input values. To compute the output function for an input that is not listed, a linear approximation is made based on the nearest inputs and nearest outputs. As an example, consider the table below.

If the input value was one of those listed then there would be no problem. Unfortunately the real world is never so kind. The input number will probably be 259 or something similar. If this is the case linear interpolation would provide a reasonable result. The formula is:

$$\text{Delta Out} = \frac{\text{UpperOutput-Lower Output}}{\text{Upper Input-Lower Input}} * (\text{Actual Input-Lower Input})$$

Actual Output = Lower Output + Delta Out

For the value of 259 the solution is:

$$\text{Delta Out} = \frac{900-400}{300-200} * (259-200) = \frac{500}{100} * 59 = 5 * 59 = 295$$

Actual Output = 400 + 295 = 695

To make the algorithm easier, (and therefore faster), it is appropriate to limit the range and accuracy of the function to only what is needed. It is also advantageous to make the input step (Upper Input-Lower Input) equal to a power of 2. This allows the substitution of multiple right shifts for a divide operation, thus speeding up throughput. The 8096 allows multiple arithmetic right shifts with a single instruction providing a very fast divide if the divisor is a power of two.

For the purpose of an example, a program with a 12-bit output and an 8-bit input has been written. An input step of 16 (2**4) was selected. To cover the input range 17 words are needed, 255/16 + 1 word to handle values in the last 15 bytes of input range. Although only 12 bits are required for the output, the 16-bit architecture offers no penalty for using 16 instead of 12 bits.

The program for this example, shown in Listing 3-2, uses the definitions and equates from Listing 3-1, only the additional equates and definitions are shown in the code.

Input Value	Relative Table Address	Table Value
100	0001H	100
200	0002H	400
300	0003H	900
400	0004H	1600

```
$TITLE('INTER1.APT: Interpolation routine 1')
;;;;;;;   8096 Assembly code for table lookup and interpolation

$INCLUDE(:F1:DEMO96.INC)        ; Include demo definitions


RSEG   at 22H

        IN_VAL:         dsb     1               ; Actual Input Value
        TABLE_LOW:      dsw     1
        TABLE_HIGH:     dsw     1
        IN_DIF:         dsw     1               ; Upper Input - Lower Input
        IN_DIFB         equ     IN_DIF  :byte
        TAB_DIF:        dsw     1               ; Upper Output - Lower Output
        OUT:            dsw     1
        RESULT:         dsw     1
        OUT_DIF:        dsl     1               ; Delta Out


CSEG at 2080H

        LD      SP, #100H
```

270061-17

Listing 3-2. ASM-96 Code for Table Lookup Routine 1

```
look:   LDB     AL, IN_VAL      ; Load temp with Actual Value
        SHRB    AL, #3          ; Divide the byte by 8
        ANDB    AL, #11111110B  ; Insure AL is a word address
                                ;  This effectively divides AL by 2
                                ;  so AL = IN_VAL/16

        LDBZE   AX, AL          ; Load byte AL to word AX
        LD      TABLE_LOW, TABLE [AX]     ; TABLE_LOW is loaded with the value
                                          ;  in the table at table location AX

        LD      TABLE_HIGH, (TABLE+2)[AX]   ; TABLE_HIGH is loaded with the
                                            ; value in the table at table
                                            ; location AX+2
                                            ; (The next value in the table)

        SUB     TAB_DIF, TABLE_HIGH, TABLE_LOW
                                            ; TAB_DIF=TABLE_HIGH-TABLE_LOW

        ANDB    IN_DIFB, IN_VAL, #0FH       ; IN_DIFB=least significant 4 bits
                                            ;  of IN_VAL
        LDBZE   IN_DIF, IN_DIFB             ; Load byte IN_DIFB to word IN_DIF

        MUL     OUT_DIF, IN_DIF, TAB_DIF
                                            ; Output_difference =
                                            ;  Input_difference*Table_difference
        SHRAL   OUT_DIF, #4                 ; Divide by 16 (2**4)

        ADD     OUT, OUT_DIF, TABLE_LOW     ; Add output difference to output
                                            ;  generated with truncated IN_VAL
                                            ;  as input
        SHRA    OUT, #4                     ; Round to 12-bit answer

        ADDC    OUT, zero                   ; Round up if Carry = 1

no_inc: ST      OUT, RESULT                 ; Store OUT to RESULT

        BR      look                        ; Branch to "look:"


cseg    AT 2100H

table:  DCW     0000H, 2000H, 3400H, 4C00H  ; A random function
        DCW     5D00H, 6A00H, 7200H, 7800H
        DCW     7B00H, 7D00H, 7600H, 6D00H
        DCW     5D00H, 4B00H, 3400H, 2200H
        DCW     1000H

END
```

270061-18

Listing 3-2. ASM-96 Code for Table Lookup Routine 1 (Continued)

If the function is known at the time of writing the software it is also possible to calculate in advance the change in the output function for a given change in the input. This method can save a divide and a few other instructions at the expense of doubling the size of the lookup table. There are many applications where time is critical and code space is overly abundant. In these cases the code in Listing 3-3 will work to the same specifications as the previous example.

```
$TITLE('INTER2.APT: Interpolation routine 2')

;;;;;;;   8096 Assembly code for table lookup and interpolation
;;;;;;;   Using tabled values in place of division

$INCLUDE(:F1:DEMO96.INC) ; Include demo definitions


RSEG   at 24H

        IN_VAL:         dsb     1           ; Actual Input Value
        TABLE_LOW:      dsw     1           ; Table value for function
        TABLE_INC:      dsw     1           ; Incremental change in function
        IN_DIF:         dsw     1           ; Upper Input - Lower Input
        IN_DIFB         equ     IN_DIF  :byte
        OUT:            dsw     1
        RESULT:         dsw     1
        OUT_DIF:        dsl     1           ; Delta Out
```

270061-19

Listing 3-3. ASM-96 Code For Table Lookup Routine 2

```
CSEG at 2080H

        LD      SP, #100H       ; Initialize SP to top of reg. file

look:   LDB     AL, IN_VAL      ; Load temp with Actual Value
        SHRB    AL, #3          ; Divide the byte by 8
        ANDB    AL, #11111110B  ; Ensure AL is a word address
                                ;  This effectively divides AL by 2
                                ;  so AL = IN_VAL/16
        LDBZE   AX, AL          ; Load byte AL to word AX

        LD      TABLE_LOW, VAL_TABLE[AX] ; TABLE_LOW is loaded with the value
                                         ;  in the value table at location AX

        LD      TABLE_INC, INC_TABLE[AX] ; TABLE_INC is loaded with the value
                                         ; in the increment table at
                                         ; location AX

        ANDB    IN_DIFB, IN_VAL, #0FH    ; IN_DIFB=least significant 4 bits
                                         ;  of IN_VAL
        LDBZE   IN_DIF, IN_DIFB          ; Load byte IN_DIFB to word IN_DIF

        MUL     OUT_DIF, IN_DIF, TABLE_INC
                                         ; Output_difference =
                                         ;  Input_difference*Incremental_change

        ADD     OUT, OUT_DIF, TABLE_LOW ; Add output difference to output
                                         ;  generated with truncated IN_VAL
                                         ;  as input
        SHR     OUT, #4                  ; Round to 12-bit answer
        ADDC    OUT, zero                ; Round up if Carry = 1

no_inc: ST      OUT, RESULT              ; Store OUT to RESULT
        BR      look                     ; Branch to "look:"

cseg    AT 2100H

val_table:
        DCW     0000H, 2000H, 3400H, 4C00H   ; A random function
        DCW     5D00H, 6A00H, 7200H, 7800H
        DCW     7B00H, 7D00H, 7600H, 6D00H
        DCW     5D00H, 4B00H, 3400H, 2200H
        DCW     1000H
inc_table:
        DCW      0200H,  0140H,  0180H,  0110H   ; Table of incremental
        DCW      00D0H,  0080H,  0060H,  0030H   ; differences
        DCW     00020H, 0FF90H, 0FF70H, 0FF00H
        DCW     0FEE0H, 0FE90H, 0FEE0H, 0FEE0H

END
```

270061-20

Listing 3-3. ASM-96 Code for Table Lookup Routine 2 (Continued)

By making use of the second lookup table, one word of RAM was saved and 16 state times. In most cases this time savings would not make much of a difference, but when pushing the processor to the limit, microseconds can make or break a design.

3.1.2. PL/M-96

Intel provides high level language support for most of its micro processors and microcontrollers in the form of PL/M. Specifically, PL/M refers to a family of languages, each similar in syntax, but specialized for the device for which it generates code. The PL/M syntax is similar to PL/1, and is easy to learn. PLM-96 is the version of PL/M used for the 8096. It is very code efficient as it was written specifically for the MCS-96 family. PLM-96 most closely resembles PLM-86, although it has bit and I/O functions similar to PLM-51. One line of PL/M-code can take the place of many lines of assembly code. This is advantageous to the programmer, since code can usually be written at a set number of lines per hour, so the less lines of code that need to be written, the faster the task can be completed.

If the first example of interpolation is considered, the PLM-96 code would be written as shown in Listing 3-4. Note that version 1.0 of PLM-96 does not support 32-bit results of 16 by 16 multiplies, so the ASM-96 procedure "DMPY" is used. Procedure DMPY, shown in Listing 3-5, must be assembled and linked with the compiled PLM-96 program using RL-96, the relocator and linker. The command line to be used is:

```
RL96 PLMEX1.OBJ, DMPY.OBJ, PLM96.LIB &
to PLMOUT.OBJ ROM (2080H-3FFFH)
```

```
/* PLM-96 CODE FOR TABLE LOOK-UP AND INTERPOLATION */

PLMEX:    DO;

DECLARE IN_VAL          WORD        PUBLIC;
DECLARE TABLE_LOW       INTEGER     PUBLIC;
DECLARE TABLE_HIGH      INTEGER     PUBLIC;
DECLARE TABLE_DIF       INTEGER     PUBLIC;
DECLARE OUT             INTEGER     PUBLIC;
DECLARE RESULT          INTEGER     PUBLIC;
DECLARE OUT_DIF         LONGINT     PUBLIC;
DECLARE TEMP            WORD        PUBLIC;

DECLARE TABLE(17)       INTEGER  DATA (
        0000H, 2000H, 3400H, 4C00H,       /* A random function */
        5D00H, 6A00H, 7200H, 7800H,
        7B00H, 7D00H, 7600H, 6D00H,
        5D00H, 4B00H, 3400H, 2200H,
        1000H);

DMPY:   PROCEDURE (A,B) LONGINT EXTERNAL;
        DECLARE (A,B) INTEGER;
END DMPY;

LOOP:
    TEMP=SHR(IN_VAL,4);      /* TEMP is the most significant 4 bits of IN_VAL */

    TABLE_LOW=TABLE(TEMP);      /* If "TEMP" was replaced by "SHR(IN_VAL,4)"  */
    TABLE_HIGH=TABLE(TEMP+1);   /* The code would work but the 8096 would     */
                                /* do two shifts                              */

    TABLE_DIF=TABLE_HIGH-TABLE_LOW;

    OUT_DIF=DMPY(TABLE_DIF,SIGNED(IN_VAL AND 0FH)) /16;

    OUT=SAR((TABLE_LOW+OUT_DIF),4); /* SAR performs an arithmetic right shift,
                                       in this case 4 places are shifted      */

    IF CARRY=0 THEN RESULT=OUT;  /* Using the hardware flags must be done     */
      ELSE  RESULT=OUT+1;        /* with care to ensure the flag is tested    */
                                 /* in the desired instruction sequence       */
GOTO LOOP;

/* END OF PLM-96 CODE  */

END;
```

270061-21

Listing 3-4. PLM-96 Code For Table Lookup Routine 1

```
        $TITLE('MULT.APT: 16*16 multiply procedure for PLM-96')

                SP      EQU     18H:word

        rseg
                EXTRN   PLMREG  :long

        cseg
                PUBLIC  DMPY            ; Multiply two integers and return a
                                        ; longint result in AX, DX registers

        DMPY:   POP     PLMREG+4                ; Load return address
                POP     PLMREG                  ; Load one operand
                MUL     PLMREG,[SP]+    ; Load second operand and increment SP

                BR      [PLMREG+4]              ; Return to PLM code.
        END
```

270061-22

Listing 3-5. 32-Bit Result Multiply Procedure For PLM-96

Using PLM, code requires less lines, is much faster to write, and easier to maintain, but may take slightly longer to run. For this example, the assembly code generated by the PLM-96 compiler takes 56.75 microseconds to run instead of 30.75 microseconds. If PLM-96 performed the 32-bit result multiply instead of using the ASM-96 routine the PLM code would take 41.5 microseconds to run. The actual code listings are shown in Appendix A.

3.2. Using the I/O Section

3.2.1. USING THE HSI UNIT

One of the most frequent uses of the HSI is to measure the time between events. This can be used for frequency determination in lab instruments, or speed/acceleration information when connected to pulse type encoders. The code in Listing 3-6 can be used to determine the high and low times of the signals on two lines. This code can be easily expanded to 4 lines and can also be modified to work as an interrupt routine.

Frequently it is also desired to keep track of the number of events which have occurred, as well as how often they are occurring. By using a software counter this feature can be added to the above code. This code depends on the software responding to the change in line state before the line changes again. If this cannot be guaranteed then it may be necessary to use 2 HSI lines for each incoming line. In this case one HSI line would look for falling edges while the other looks for rising edges. The code in Listing 3-7 includes both the counter feature and the edge detect feature.

The uses for this type of routine are almost endless. In instrumentation it can be used to determine frequency on input lines, or perhaps baud rate for a self adjusting serial port. Section 4.2 contains an example of making a software serial port using the HSI unit. Interfacing to some form of mechanically generated position information is a very frequent use of the HSI. The applications in this category include motor control, precise positioning (print heads, disk drives, etc.), engine control and

```
$TITLE('PULSE.APT: Measuring pulses using the HSI unit')

$INCLUDE(DEMO96.INC)

rseg    at  28H

        HIGH_TIME:      dsw     1
        LOW_TIME:       dsw     1
        PERIOD:         dsw     1
        HI_EDGE:        dsw     1
        LO_EDGE:        dsw     1


cseg    at  2080H

        LD      SP, #100H
        LDB     IOC0, #00000001B        ; Enable HSI 0
        LDB     HSI_MODE, #00001111B    ; HSI 0 look for either edge

wait:   ADD     PERIOD, HIGH_TIME, LOW_TIME
        JBS     IOS1, 6, contin         ; If FIFO is full
        JBC     IOS1, 7, wait   ; Wait while no pulse is entered

contin: LDB     AL, HSI_STATUS          ; Load status; Note that reading
                                        ;   HSI_TIME clears HSI_STATUS

        LD      BX, HSI_TIME            ; Load the HSI_TIME

        JBS     AL, 1, hsi_hi           ; Jump if HSI.0 is high

hsi_lo: ST      BX, LO_EDGE
        SUB     HIGH_TIME, LO_EDGE, HI_EDGE
        BR      wait


hsi_hi: ST      BX, HI_EDGE
        SUB     LOW_TIME, HI_EDGE, LO_EDGE
        BR      wait

        END
```

270061-23

Listing 3-6. Measuring Pulses Using The HSI Unit

transmission control. The HSI unit is used extensively in the example in section 4.3.

3.2.2. USING THE HSO UNIT

Although the HSO has many uses, the best example is that of a multiple PWM output. This program, shown in Listing 3-8, is simple enough to be easily understood, yet it shows how to use the HSO for a task which can be complex. In order for this program to operate, another program needs to set up the on and off time variables for each line. The program also requires that a HSO line not change so quickly that it changes twice between consecutive reads of I/O Status Register 0, (IOS0).

A very eye catching example can be made by having the program output waveforms that vary over time. The driver routine in Listing 3-10 can be linked to the above program to provide this function. Linking is accomplished using RL96, the relocatable linker for the 8096. Information for using RL96 can be found in the "MCS-96 Utilities Users Guide", listed in the bibliography. In order for the program to link, the register dec-

```
$TITLE ('ENHSI.APT: ENHANCED HSI PULSE ROUTINE')

$INCLUDE(DEMO96.INC)

RSEG AT 28H

        TIME:           DSW 1
        LAST_RISE:      DSW 1
        LAST_FALL:      DSW 1
        HSI_S0:         DSB 1
        IOS1_BAK:       DSB 1
        PERIOD:         DSW 1
        LOW_TIME:       DSW 1
        HIGH_TIME:      DSW 1
        COUNT:          DSW 1

cseg    at      2080H

init:   LD      SP,#100H

        LDB     IOC1,#00100101B ; Disable HSO.4,HSO.5, HSI_INT=first,
                                ; Enable PWM,TXD,TIMER1_OVRFLOW_INT

        LDB     HSI_MODE,#10011001B     ; set hsi.1 -; hsi.0 +
        LDB     IOC0,#00000111B         ; Enable  hsi 0,1
                                        ; T2 CLOCK=T2CLK, T2RST=T2RST
                                        ; Clear timer2


wait:   ANDB    IOS1_BAK,#01111111B     ; Clear IOS1_BAK.7
        ORB     IOS1_BAK,IOS1           ; Store into temp to avoid clearing
                                        ; other flags which may be needed
        JBC     IOS1_BAK,7,wait         ; If hsi is not triggered then
                                        ; jump to wait

        ANDB    HSI_S0,HSI_STATUS,#01010101B
        LD      TIME, HSI_TIME

        JBS     HSI_S0,0,a_rise
        JBS     HSI_S0,2,a_fall
        BR      no_cnt

a_rise: SUB     LOW_TIME, TIME, LAST_FALL
        SUB     PERIOD, TIME,LAST_RISE
        LD      LAST_RISE, TIME
        BR      increment

a_fall: SUB     HIGH_TIME, TIME, LAST_RISE
        SUB     PERIOD, TIME,LAST_FALL
        LD      LAST_FALL, TIME

increment:
        INC     COUNT

no_cnt: BR      wait

        END
```

270061-24

Listing 3-7. Enhanced HSI Pulse Measurement Routine

```
$TITLE ('HSOPWM.APT: 8096 EXAMPLE PROGRAM FOR PWM OUTPUTS')

; This program will provide 3 PWM outputs on HSO pins 0-2
; The input parameters passed to the program are:
;
;               HSO_ON_N     HSO on time for pin N
;               HSO_OFF_N    HSO off time for pin N

;       Where:  Times are in timer1 cycles
;               N takes values from 0 to 3

;;;
```

laration section (i.e., the section between "RSEG" and "CSEG") in Listing 3-8 must be changed to that in Listing 3-9.

The driver routine simply changes the duty cycle of the waveform and sets the second HSO output to a frequency twice that of the first one. A slightly different driver routine could easily be the basis for a switching power supply or a variable frequency/variable voltage motor driver. The listing of the driver routine is shown in Listing 3-10.

```
;     NOTE: Use this file to replace the declaration section of
;           the HSO PWM program from "$INCLUDE(DEMO96.INC)" through
;           the line prior to the label "wait".  Also change the last
;           branch in the program to a "RET".

RSEG

          D_STAT:          DSB      1
          extrn    HSO_ON_0 :word , HSO_OFF_0 :word
          extrn    HSO_ON_1 :word , HSO_OFF_1 :word
          extrn    HSO_TIME :word , HSO_COMMAND :byte
          extrn    TIMER1   :word , IOS0       :byte
          extrn    SP       :word

          public   OLD_STAT
          OLD_STAT:         dsb      1
          NEW_STAT:         dsb      1


cseg
          PUBLIC   wait
```

270061-26

Listing 3-9. Changes to Declarations for HSO Routine

```
          $TITLE('HSODRV.APT: Driver module for HSO PWM program')

          HSODRV           MODULE  MAIN, STACKSIZE(8)


                   PUBLIC   HSO_ON_0 , HSO_OFF_0
                   PUBLIC   HSO_ON_1 , HSO_OFF_1
                   PUBLIC   HSO_TIME , HSO_COMMAND
                   PUBLIC   SP , TIMER1 , IOS0

          $INCLUDE(DEMO96.INC)

          rseg at 28H

                   EXTRN    OLD_STAT          :byte

                   HSO_ON_0:         dsw      1
                   HSO_OFF_0:        dsw      1
                   HSO_ON_1:         dsw      1
                   HSO_OFF_1:        dsw      1
                   count:            dsb      1

          cseg  at 2080H

                   EXTRN    wait      :entry

          strt:    DI
                   LD       SP, #100H
                   ANDB     OLD_STAT, IOS0, #0FH
                   XORB     OLD_STAT, #0FH

          initial:
                   LD       CX, #0100H

          loop:    LD       AX, #1000H
                   SUB      BX, AX, CX
                   LD       AX, CX

                   ST       AX, HSO_ON_0
                   ST       BX, HSO_OFF_0
```

270061-27

Listing 3-10. Driver Module for HSO PWM Program

```
        SHR     AX,#1
        SHR     BX,#1
        ST      AX, HSO_ON_1
        ST      BX, HSO_OFF_1

        CALL    wait

        INC     CX
        CMP     CX, #00F00H
        BNE     loop

        BR      initial

        END
```

270061-28

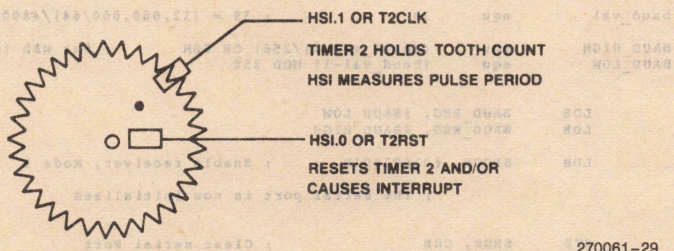
Listing 3-10. Driver Module for HSO PWM Program (Continued)

Since the 8096 needs to keep track of events which often repeat at set intervals it is convenient to be able to have Timer 2 act as a programmable modulo counter. There are several ways of doing this. The first is to program the HSO to reset Timer 2 when Timer 2 equals a set value. A software timer set to interrupt at Timer 2 equals zero could be used to reload the CAM. This software method takes up two locations in the CAM and does not synchronize Timer 2 to the external world.

To synchronize Timer 2 externally the T2 RST (Timer 2 ReSeT) pin can be used. In this way Timer 2 will get reset on each rising edge of T2 RST. If it is desired to have an interrupt generated and time recorded when Timer 2 gets reset, the signal for its reset can be taken from HSI.0 instead of T2RST. The HSI.0 pin has its own interrupt vector which functions independently of the HSI unit.

Another option available is to use the HSI.1 pin to clock Timer 2. By using this approach it is possible to use the HSI to measure the period of events on the input to Timer 2. If both of the HSI pins are used instead of the T2RST and T2CLK pins the HSIO unit can keep track of speed and position of the rotating device with very little software overhead. This type of setup is ideal for a system like the one shown in Figure 3-1, and similar to the one used in section 4.3.

In this system a sequence of events is required based on the position of the gear which represents any piece of rotating machinery. Timer 2 holds the count of the number of tooth edges passed since the index mark. By using HSI.1 as the input to Timer 2, instead of T2 CLK, it is possible to determine tooth count and time information through the HSI. From this information instantaneous velocity and acceleration can be calculated. Having the tooth edge count in Timer 2 means



270061-29

Figure 3-1. Using the HSIO to Monitor Rotating Machinery

that the HSO unit can be used to initiate the desired tasks at the appropriate tooth count. The interrupt routine initiated by HSI.0 can be used to perform any software task required every revolution. In this system, the overhead which would normally require extensive software has been done with the hardware on the 8096, thus making more software time available for control programs.

3.2.3. USING THE SERIAL PORT IN MODE 1

Mode 1 of the serial port supports the basic asynchronous 8-bit protocol and is used to interface to most CRTs and printers. The example in Listing 3-11 shows a simple routine which receives a character and then transmits the same character. The code is set up so that minor modifications could make it run on an interrupt basis. Note that it is necessary to set up some flags as initial conditions to get the routine to run properly. If it was desired to send 7 bits of data plus parity instead of 8 bits of data the PEN bit would be set to a one. Interprocessor communication, as described in section 2.3.4, can be set up by simply adding code to change RB8 and the port mode to the listing below. The hardware shown in Figure 3-2 can be used to convert the logic level output of the 8096 to ±12 or 15 volt levels to connect to a CRT. This circuit has been found to work with most RS-232 devices, although it does not conform to strict RS-232 specifications. If true RS-232 conformance is required then any standard RS-232 driver can be used.

```
$TITLE('SP.APT: SERIAL PORT DEMO PROGRAM')

$INCLUDE(DEMO96.INC)

rseg    at 28H

        CHR:    dsb     1
        SPTEMP: dsb     1
        TEMP0:  dsb     1
        TEMP1:  dsb     1
        RCV_FLAG:       dsb     1

cseg    at 200CH

        DCW     ser_port_int

cseg    at 2080H

        LD      SP, #100H

        LDB     IOC1, #00100000B                ; Set P2.0 to TXD

                ; Baud rate = input frequency / (64*baud_val)
                ; baud_val  = (input frequency/64) / baud rate


baud_val        equ     39              ; 39 = (12,000,000/64)/4800 baud

BAUD_HIGH       equ     ((baud_val-1)/256) OR 80H       ; Set MSB to 1
BAUD_LOW        equ     (baud_val-1) MOD 256


        LDB     BAUD_REG, #BAUD_LOW
        LDB     BAUD_REG, #BAUD_HIGH

        LDB     SPCON, #01001001B       ; Enable receiver, Mode 1

                ; The serial port is now initialized


        STB     SBUF, CHR               ; Clear serial Port
        LDB     TEMP0, #00100000B       ; Set TI-temp

        LDB     INT_MASK, #01000000B    ; Enable Serial Port Interrupt
        EI
loop:   BR      loop            ; Wait for serial port interrupt


ser_port_int:
        PUSHF
rd_again:                               ; This section of code can be replaced
        LDB     SPTEMP, SPSTAT          ; with "ORB  TEMP0, SP_STAT" when the
        ORB     TEMP0, SPTEMP           ; serial port TI and RI bugs are fixed
        ANDB    SPTEMP,#11100000B
        JNE     rd_again        ; Repeat until TI and RI are properly cleared
```

270061-30

Listing 3-11. Using the Serial Port in Mode 1

```
get_byte:
          JBC     TEMP0, 6, put_byte      ; If RI-temp is not set
          STB     SBUF, CHR               ; Store byte
          ANDB    TEMP0, #10111111B       ; CLR RI-temp
          LDB     RCV_FLAG, #0FFH         ; Set bit-received flag

put_byte:
          JBC     RCV_FLAG, 0, continue   ; If receive flag is cleared
          JBC     TEMP0, 5, continue      ; If TI was not set
          LDB     SBUF, CHR               ; Send byte
          ANDB    TEMP0, #11011111B       ; CLR TI-temp

          ANDB    CHR, #01111111B         ; This section of code appends
          CMPB    CHR,#0DH                ; an LF after a CR is sent
          JNE     clr_rcv
          LDB     CHR, #0AH
          BR      continue
clr_rcv:
          CLRB    RCV_FLAG                ; Clear bit-received flag

continue:
          POPF
          RET

          END
```

270061-31

Listing 3-11. Using the Serial Port in Mode 1 (Continued)

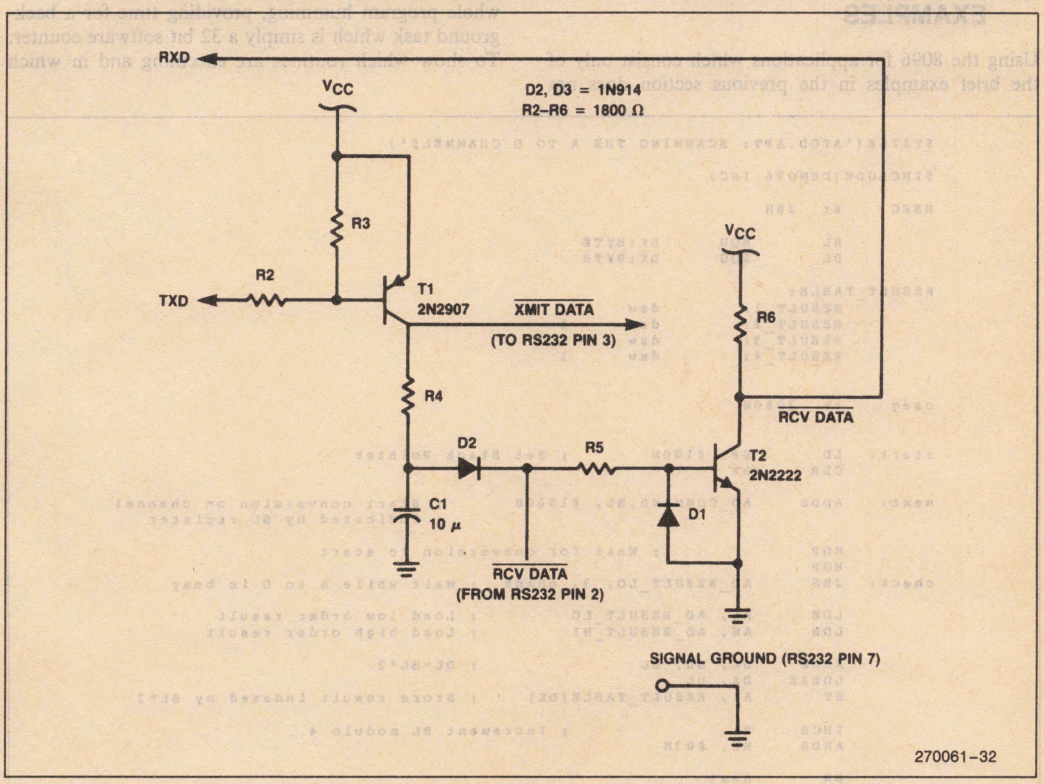


Figure 3-2. Serial Port Level Conversion

3.2.4. USING THE A TO D

The code in Listing 3-12 makes use of the software flags to implement a non-interrupt driven routine which scans A to D channels 0 through 3 and stores them as words in RAM. An interrupt driven routine is shown in section 4.1. When using the A to D it is important to always read the value using the byte read commands, and to give the converter 8 state times to start converting before reading the status bit.

Since there is no sample and hold on the A to D converter it may be desirable to use an RC filter on each input. A 100Ω resistor in series with a 0.22 uf capacitor to ground has been used successfully in the lab. This circuit gives a time constant of around 22 microseconds which should be long enough to get rid of most noise, without overly slowing the A to D response time.

4.0 ADVANCED SOFTWARE EXAMPLES

Using the 8096 for applications which consist only of the brief examples in the previous section does not really make use of its full capabilities. The following examples use some of the code blocks from the previous section to show how several I/O features can be used together to accomplish a practical task. Three examples will be shown. The first is simply a combination of several of the section 3 examples run under an interrupt system. Next, a software serial port using the HSIO unit is described. The concluding example is one of interfacing the HSI unit to an optical encoder to control a motor.

4.1. Simultaneous I/O Routines under Interrupt Control

A four channel analog to PWM converter can easily be made using the 8096. In the example in Listing 4 analog channels are read and 3 PWM waveforms are generated on the HSO lines and one on the PWM pin. Each analog channel is used to set the duty cycle of its associated output pin. The interrupt system keeps the whole program humming, providing time for a background task which is simply a 32 bit software counter. To show which routines are executing and in which

```
$TITLE('ATOD.APT: SCANNING THE A TO D CHANNELS')

$INCLUDE(DEMO96.INC)

RSEG    at  28H

        BL      EQU     BX:BYTE
        DL      EQU     DX:BYTE

RESULT_TABLE:
        RESULT_1:       dsw     1
        RESULT_2:       dsw     1
        RESULT_3:       dsw     1
        RESULT_4:       dsw     1


cseg    at  2080H


start:  LD      SP, #100H       ; Set Stack Pointer
        CLR     BX

next:   ADDB    AD_COMMAND,BL, #1000B       ; Start conversion on channel
                                            ; indicated by BL register

        NOP             ; Wait for conversion to start
        NOP
check:  JBS     AD_RESULT_LO, 3, check  ; Wait while A to D is busy

        LDB     AL, AD_RESULT_LO        ; Load low order result
        LDB     AH, AD_RESULT_HI        ; Load high order result

        ADDB    DL, BL, BL              ; DL=BL*2
        LDBZE   DX, DL
        ST      AX, RESULT_TABLE[DX]    ; Store result indexed by BL*2

        INCB    BL              ; Increment BL modulo 4
        ANDB    BL, #03H

        BR      next

        END
```

270061-33

Listing 3-12. Scanning the A to D Channels

order, Port 1 output pins are used to indicate the current status of each task. The actual code listing is included in Appendix B.

The initialization section, shown in Listing 4-1a, clears a few variables and then loads the first set of on and off times to the HSO unit. Note that 8 state times must be waited between consecutive loads of the HSO. If this is not done it is possible to overwrite the contents of the CAM holding register. An A/D interrupt is forced by setting the bit in the Interrupt Pending register. This causes the first A/D interrupt to occur just after the Interrupt Mask register is set and interrupts are enabled.

Listing 4-1. Using Multiple I/O Devices

```
$TITLE ('8096 EXAMPLE PROGRAM FOR PWM OUTPUTS FROM A TO D INPUTS')
$PAGEWIDTH(130)

; This program will provide 3 PWM outputs on HSO pins 0-2
; and one on the PWM.
;
; The PWM values are determined by the input to the A/D converter.
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

$INCLUDE(DEMO96.INC)

RSEG AT 28H

        DL      EQU     DX:BYTE

ON_TIME:
        PWM_TIME_1:     DSW     1
        HSO_ON_0:       DSW     1
        HSO_ON_1:       DSW     1
        HSO_ON_2:       DSW     1

RESULT_TABLE:
        RESULT_0:       DSW     1
        RESULT_1:       DSW     1
        RESULT_2:       DSW     1
        RESULT_3:       DSW     1

        NXT_ON_T:       DSW     1
        NXT_OFF_0:      DSW     1
        NXT_OFF_1:      DSW     1
        NXT_OFF_2:      DSW     1
        COUNT:          DSL     1
        AD_NUM:         DSW     1       ; Channel being converted
        TMP:            DSW     1
        HSO_PER:        DSW     1
        LAST_LOAD:      DSB     1

cseg    AT 2000H

        DCW     start           ; Timer_ovf_int
        DCW     Atod_done_int
        DCW     start           ; HSI_data_int
        DCW     HSO_exec_int


cseg    AT 2080H

start:  LD      SP, #100H       ; Set Stack Pointer
        CLR     AX
wait:   DEC     AX              ; wait approx. 0.2 seconds for
        JNE     wait            ; SBE to finish communications

        CLRB    AD_NUM

        LD      PWM_TIME_1, #080H
        LD      HSO_PER, #100H
        LD      HSO_ON_0, #040H
        LD      HSO_ON_1, #080H
        LD      HSO_ON_2, #0C0H

        ADD     NXT_ON_T,  Timer1, #100H
```

270061-34

Listing 4-1a. Initializing the A to D to PWM Program

```
        LDB     HSO_COMMAND, #00110110B     ; Set HSO for timer1, set pin 0,1
        LD      HSO_TIME, NXT_ON_T          ; with interrupt
        NOP
        NOP
        LDB     HSO_COMMAND, #00100010B     ; Set HSO for timer1, set pin 2
        ADD     HSO_TIME, NXT_ON_T          ; without interrupt

        ORB     LAST_LOAD, #00000111B   ; Last loaded value was set all pins
        LDB     INT_MASK, #00001010B    ; Enable HSO and A/D interrupts
        LDB     INT_PENDING, #00001010B ; Fake an A/D and HSO interrupt
        EI

loop:   ORB     Port1, #00000001B       ; set P1.0
        ADD     COUNT, #01
        ADDC    COUNT+2,zero
        ANDB    Port1, #11111110B       ; clear P1.0
        BR      loop
```

270061-35

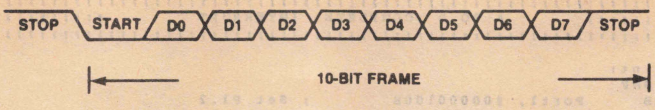
Listing 4-1a. Initializing the A to D to PWM program (Continued)

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;          HSO  EXECUTED  INTERRUPT          ;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

HSO_exec_int:
        PUSHF
        ORB     Port1, #00000010B       ; Set p1.1

        SUB     TMP,TIMER1, NXT_ON_T
        CMP     TMP,ZERO
        JLT     set_off_times

set_on_times:
        ADD     NXT_ON_T, HSO_PER
        LDB     HSO_COMMAND, #00110110B   ; Set HSO for timer1, set pin 0,1
        LD      HSO_TIME, NXT_ON_T
        NOP
        NOP
        LDB     HSO_COMMAND, #00100010B   ; Set HSO for timer1, set pin 2
        LD      HSO_TIME, NXT_ON_T

        ORB     LAST_LOAD, #00000111B     ; Last loaded value was all ones

        LDB     PWM_CONTROL, PWM_TIME_1       ; Now is as good a time as any
                                              ; to update the PWM reg
        BR      check_done


set_off_times:
        JBC     LAST_LOAD, 0, check_done

        ADD     NXT_OFF_0, NXT_ON_T, HSO_ON_0
        LDB     HSO_COMMAND, #00010000B       ; Set HSO for timer1, clear pin 0
        LD      HSO_TIME, NXT_OFF_0

        NOP
        ADD     NXT_OFF_1, NXT_ON_T, HSO_ON_1
        LDB     HSO_COMMAND, #00010001B       ; Set HSO for timer1, clear pin 1
        LD      HSO_TIME, NXT_OFF_1

        NOP
        ADD     NXT_OFF_2, NXT_ON_T, HSO_ON_2
        LDB     HSO_COMMAND, #00010010B       ; Set HSO for timer1, clear pin 2
        LD      HSO_TIME, NXT_OFF_2

        ANDB    LAST_LOAD, #11111000B   ; Last loaded value was all 0s

check_done:
        ANDB    Port1, #11111101B       ; Clear P1.1
        POPF
        RET
```

270061-36

Listing 4-1b. Interrupt Driven HSO Routine

```
;;;
```



270061-38

Figure 4-1. 10-bit Asynchronous Frame

antee that the leading edge of the START bit will cause a transition on the line; it also provides for a dead time on the line so that the receiver can maintain its synchronization.

The remainder of this section will show how a full-duplex asynchronous port can be built from the HSIO unit. There are four sections to this code:

1. Interface routines. These routines provide a procedural interface between the interrupt driven core of the software serial port and the remainder of the application software.
2. Initialization routine. This routine is called during the initialization of the overall system and sets up the various variables used by the software port.
3. Transmit ISR. This routine runs as an ISR (interrupt service routine) in response to an HSO interrupt interrupt. Its function is to serialize the data passed to it by the interface routines.
4. Receive ISRs. There are two ISRs involved in the receive process. One of them runs in response to an HSI interrupt and is used to synchronize the receive process at the leading edge of the start bit. The second receive ISR runs in response to an HSO generated software timer interrupt, this routine is scheduled to run at the center of each bit and is used to deserialize the incoming data.

The routines share the set of variables that are shown in Listing 4-2. These variables should be accessed only by the routines which make up the software serial port.

```
;
;       VARIABLES NEEDED BY THE SOFTWARE SERIAL PORT
;       ============================================
;
        rseg

rcve_state:     dsb 1
  rxrdy         equ 1           ; indicates receive done
  rxoverrun     equ 2           ; indicates receive overflow
  rip           equ 4           ; receive in progress flag
rcve_buf:       dsb 1           ; used to double buffer receive data
rcve_reg:       dsb 1           ; used to deserialize receive
sample_time:    dsw 1           ; records last receive sample time

serial_out:     dsw 1           ; Holds the output character+framing (start and
                                ;   stop bits) for transmit process.
baud_count:     dsw 1           ; Holds the period of one bit in units
                                ;   of T1 ticks.
txd_time:       dsw 1           ; Transition time of last Txd bit that was
                                ;   sent to the CAM
char:           dsb 1           ; for test only
;
;       COMMANDS ISSUED TO THE HSO UNIT
;       ===============================
;
mark_command    equ     01101101b       ; timer1,set,interrupt on 5
space_command   equ     00101101b       ; timer1,clr,interrupt on 5
sample_command  equ     00110000b       ; software timer 0

$eject
```

270061-39

Listing 4-2. Software Serial Port Declarations

The table also shows the declarations for the commands issued to the HSO unit. In this example HSI.2 is used for receive data and HSO.5 is used for transmit data, although other HSI and HSO lines could have been used.

The interface routines are shown in Listing 4-3. Data is passed to the port by pushing the eight-bit character into the stack and calling *char_out*, which waits for any in-process transmission to complete and stores the character into the variable *serial_out*. As the data is stored the START and STOP bits are added to the data bits. The routine *char—in* is called when the application software requires a character from the port. The data is returned in the *ax* register in conformance to PLM 96 calling conventions. The routine *csts* can be called to determine if a character is available at the port before calling *char_in*. (If no character is available *char_in* will wait indefinitely).

The initialization routine is shown in Listing 4-4. This routine is called with the required baud rate in the

```
;
char_out:
; Output character to the software serial port
;
        pop     cx                  ; the return address
        pop     bx                  ; the character for output
        ldb     (bx+1),#01h         ; add the start and stop bits
        add     bx,bx               ;   to the char and leave as 16 bit
wait_for_xmit:
        cmp     serial_out,0        ; wait for serial_out=0 (it will be cleared by
        bne     wait_for_xmit       ;   the hso interrupt process)
        st      bx,serial_out       ; put the formatted character in serial_out
        br      [cx]                ; return to caller
;
csts:
; Returns "true" (ax<>0) if char_in has a character.
;
        clr     ax
        bbc     rcve_state,0,csts_exit
        inc     ax
csts_exit:
        ret
;
char_in:
; Get a character from the software serial port
;
                                    ; wait for character ready
        bbc     rcve_state,0,char_in
        pushf                       ; set up a critical region
        andb    rcve_state,#not(rxrdy)
        ldbze   al,rcve_buf
        popf                        ; leave the critical region
        ret
```

270061-40

Listing 4-3. Software Serial Port Interface Routines

```
;
setup_serial_port:
; Called on system reset to intiate the software serial port.
;
        pop     cx                  ; the return address
        pop     bx                  ; the baud rate (in decimal)
        ld      dx,#0007h           ; dx:ax:=500,000 (assumes 12 Mhz crystal)
        ld      ax,#0A120h
        divu    ax,bx               ; calculate the baud count (500,000/baudrate)
        st      ax,baud_count
        st      0,serial_out        ; clear serial out
        ldb     ioc1,#01100000b     ; Enable HSO.5 and Txd
        bbs     ios0,6,$            ; Wait for room in the HSO CAM
                                    ; and issue a MARK command.
        add     txd_time,timer1,20
        ldb     hso_command,#mark_command
        ld      hso_time,txd_time
        clrb    rcve_buf            ; clear out the receive variables
        clrb    rcve_reg
        clrb    rcve_state
        call    init_receive        ; setup to detect a start bit
        br      [cx]                ; return
```

270061-41

Listing 4-4. Software Serial Port Initialization Routine

stack; it calculates the bit time from the baud rate and stores it in the variable *baud_count* in units of TIMER1 ticks. An HSO command is issued which will initiate the transmit process and then the remainder of the variables owned by the port are initialized. The routine *init_receive* is called to setup the HSI unit to look for the leading edge of the START bit.

The transmit process is shown in Listing 4-5. The HSO unit is used to generate an output command to the transmit pin once per bit time. If the *serial_out* register is zero a MARK (idle condition) is output. If the *serial_out* register contains data then the least significant bit is output and the register shifted right one place. The framing information (START and STOP bits) are appended to the actual data by the interface routines. Note that this routine will be executed once per bit time whether or not data is being transmitted. It would be possible to use this routine for additional low resolution timing functions with minimal overhead.

The receive process consists of an initialization routine and two interrupt service routines, *hsi_isr* and *software_timer_isr*. The listings of these routines are shown in Listings 4-6a,4-6b, and 4-6c respectively. The

```
;
hso_isr:
; Fields the hso interrupts and performs the serialization of the data.
; Note: this routine would be incorporated into the hso service strategy for an
;       actual system.

        cseg    at 2006h
        dcw     hso_isr             ; Set up vector

        cseg
        pushf
        add     txd_time,baud_count
        cmp     serial_out,0        ; if character is done send a mark
        be      send_mark
        shr     serial_out,#1       ; else send bit 0 of serial_out and shift
        bc      send_mark           ;   serial_out left one place.
send_space:
        ldb     hso_command,#space_command
        ld      hso_time,txd_time
        br      hso_isr_exit
send_mark:
        ldb     hso_command,#mark_command
        ld      hso_time,txd_time

hso_isr_exit:
        popf
        ret
$eject
```

270061-42

Listing 4-5. Software Serial Port Transmit Process

Listing 4-6. Receive Process

```
;
init_receive:
; Called to prepare the serial input process to find the leading edge of
; a start bit.
;
        ldb     ioc0,#00000000b         ; disconnect change detector
        ldb     hsi_mode,#00100000b     ; negative edges on HSI.2
flush_fifo:
        orb     iosl_save,iosl
        bbc     iosl_save,7,flush_fifo_done
        ldb     al,hsi_status
        ld      ax,hsi_time             ; trash the fifo entry
        andb    iosl_save,#not(80h)     ; clear bit 7.
        br      flush_fifo
flush_fifo_done:
        ldb     ioc0,#00010000b         ; connect HSI.2 to detector
        ret
```

270061-43

Listing 4-6a. Software Serial Port Receive Initialization

```
;
hsi_isr:
; Fields interrupts from the HSI unit, used to detect the leading edge
; of the START bit
; Note: this routine would be incorporated into the HSI strategy of an actual
; system.
;
        cseg at 2004h
        dcw     hsi_isr                 ; setup the interrupt vector

        cseg
        pushf
        push    ax
        ldb     al,hsi_status
        ld      sample_time,hsi_time
        bbc     al,4,exit_hsi
        bbs     ios0,7,$                ; wait for room in HSO holding reg
        ld      ax,baud_count           ; send out sample command in 1/2
        shr     ax,#1                   ; bit time
        add     sample_time,ax
        ldb     hso_command,#sample_command
        st      sample_time,hso_time
        ldb     ioc0,#00000000b         ; disconnect hsi.2 from change detector
exit_hsi:
        pop     ax
        popf
        ret
```

270061-44

Listing 4-6b. Software Serial Port Start Bit Detect

```
;
software_timer_isr:
; Fields the software timer interrupt, used to deserialize the incomming data.
; Note: this routine would be incorporated into the software timer stategy
; in an actual system.
;
        cseg at 200ah
        dcw     software_timer_isr      ; setup vector

        cseg
        pushf
        orb     ios1_save,ios1
        andb    ios1_save,#not(01h)     ; clear bit 0
        andb    0,rcve_state,#0fch      ; All bits except rxrdy and overrun=0
        bne     process_data
process_start_bit:
        bbc     hsi_status,5,start_ok
        call    init_receive
        br      software_timer_exit
start_ok:
        orb     rcve_state,#rip ; set receive in progress flag
        br      schedule_sample

process_data:
        bbs     rcve_state,7,check_stopbit
        shrb    rcve_reg,#1
        bbc     hsi_status,5,datazero
        orb     rcve_reg,#80h   ; set the new data bit
datazero:
        addb    rcve_state,#10h ; increment bit count
        br      schedule_sample

check_stopbit:
        bbc     hsi_status,5,$  ; DEBUG ONLY
        ldb     rcve_buf,rcve_reg
        orb     rcve_state,#rxrdy
        andb    rcve_state,#03h ; Clear all but ready and overrun bits
        call    init_receive
        br      software_timer_exit

schedule_sample:
        bbs     ios0,7,$        ; wait for holding reg empty
        ldb     hso_command,#sample_command
        add     sample_time,baud_count
        st      sample_time,hso_time

software_timer_exit:
        popf
        ret
```

270061-45

Listing 4-6c. Software Serial Port Data Reception

start is detected by the *hsi__isr* which schedules a software timer interrupt in one-half of a bit time. This first sample is used to verify that the START bit has not ended prematurely (a protection against a noisy line). The software timer service routine uses the variable *rcve__state* to determine whether it should check for a valid START bit, deserialize data, or check for a valid STOP bit. When a complete character has been received it is moved to the receive buffer and *init__receive* is called to set up the receive process for the next character. This routine is also called when an error (e.g., invalid START bit) is detected.

Appendix C contains the complete listing of the routines and the simple loop which was used to initialize them and verify their operation. The test was run for several hours at 9600 baud with no apparent malfunction of the port.

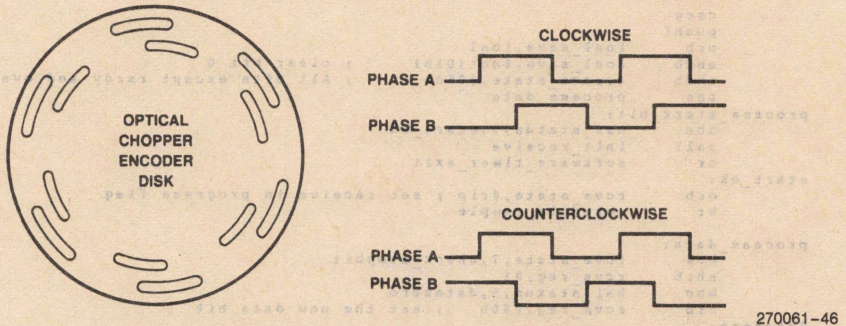
4.3. Interfacing an Optical Encoder to the HSI Unit

Optical encoders are among one of the more popular devices used to determine position of rotating equipment. These devices output two pulse trains with edges that occur from 2 to 4000 times a revolution.

Frequently there is a third line which generates one pulse per revolution for indexing purposes. Figure 4-2 shows a six line encoder and typical waveforms. As can be seen, the two waveforms provide the ability to determine both position and direction. Since a microcontroller can perform real time calculations it is possible to determine velocity and acceleration from the position and time information.

Interfacing to the encoder can be an interesting problem, as it requires connecting mechanically generated electrical signals to the HSI unit. The problems arise because it is difficult to obtain the exact nature of the signals under all conditions.

The equipment used in the lab was a Pittman 9400 series gearmotor with a 600 line optical encoder from Vernitech. The encoder has to be carefully attached to the shaft to minimize any runout or endplay. Fortunately, Pitmann has started marketing their motors with ball bearings and optical encoders already installed. It is recommended that the encoder be mounted to the motor using the exact specifications of the encoder manufacturer and/or a good machine shop.



Inside track generates Phase A. Outside track generates Phase B.

Figure 4-2. Optical Encoder and Waveforms

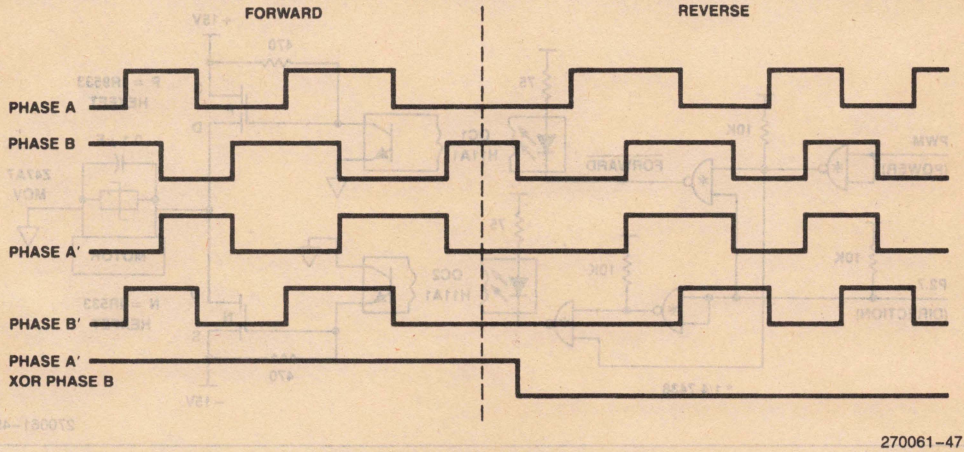
Digital filtering external to the 8096 is used on the encoder signals. The idealized signals coming from the encoder and after the digital filter are shown in Figure 4-3. The circuitry connecting the encoder to the 8096 requires only two chips. A one-shot constructed of XOR gates generates pulses on each edge of each signal. The pulses generated by Phase A are used to clock the signal from Phase B and vice versa. The hardware is shown in Figure 4-4. CMOS parts are used to reduce loading on the encoder so that buffers are not needed. Note that T2CLK is clocked on both edges of both filtered phases.

By using this method repetitive edges on a single phase without an edge on the other phase will not be passed on to the 8096. Repetitive edges on a phase can occur when the motor is stopped and vibrates or when it is changing direction. The digital filtering technique causes a little more delay in the signal at slow speeds than an analog filter would, but the simplicity trade off is worthwhile. The net effect of digital filtering is losing the ability to determine the first edge after a direction change. This does not affect the count since the first edge in both directions is lost.

If it is desired to determine when each edge occurs before filtering, the encoder outputs can be attached directly to the 8096. As these would be input signals, Port 0 is the most likely choice for connection. It would not be required to connect these lines to the HSI unit, as the information on them would only be needed when the motor is going very slowly.

The motor is driven using the PWM output pin for power control and a port pin for direction control. The 8096 drives a 7438 which drives 2 opto-isolators. These in turn drive two VFETs. A MOV (Metal Oxide Varistor, a type of transient absorber) is used to protect the VFETs, and a capacitor filters the PWM to get the best motor performance. Figure 4-5 shows the driver circuitry. To avoid noise getting into the 8096 system, the ±15 volt power supply is isolated from the 8096 logic power supply.

This is the extent of the external circuitry required for this example. All of the counting and direction detection are done by the 8096. There are two sections to the example: driving the motor and interfacing to the encoder. The motor driver uses proportional control with



NOTES:
Phase A' is Phase A clocked by Phase B
Phase B' is Phase B clocked by Phase A

Figure 4-3. Filtered Encoder Waveforms

some modifications and a braking algorithm. Since the main point of this example is I/O interfacing, the motor driver will be briefly described at the end of this section.

In order to interface to the encoder it is necessary to know the types of waveforms that can be expected. The motor was accelerated and decelerated many times using different maximum voltages. It was found that the

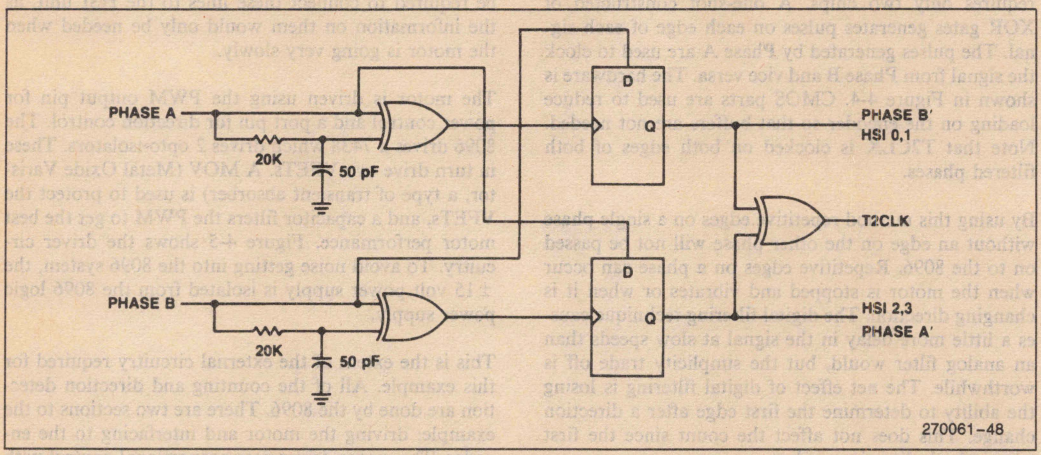


Figure 4-4. Schematic of Optical Encoder to 8096 Interface

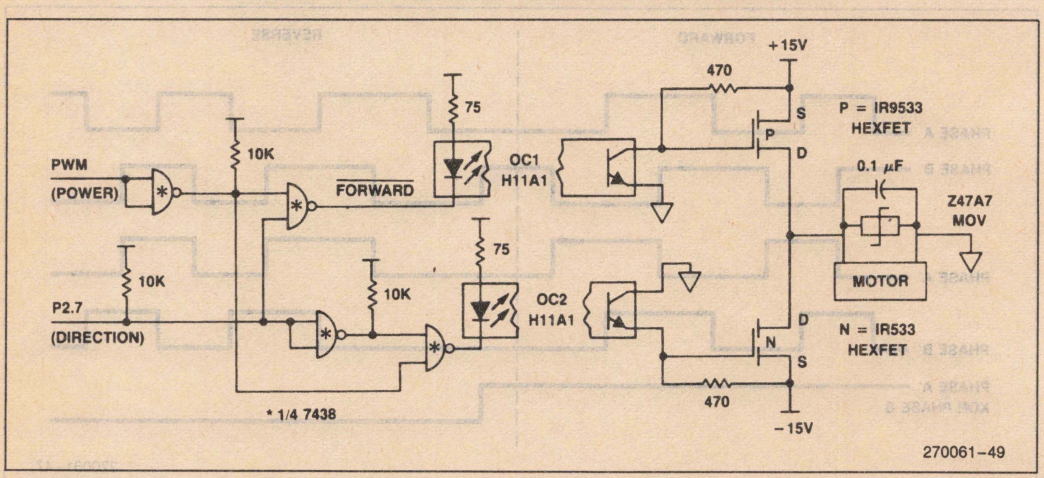


Figure 4-5. Motor Driver Circuitry

motor would decelerate smoothly until the time between encoder edges was around 100 microseconds. At this point the motor would either continue to decelerate slowly, or would suddenly stop and reverse. The latter case is the one that was most problematic.

After a brief overview, each section of the program will be described separately, with the complete listing included in the Appendix D. In order to make debugging easier, as well as to provide insight into how the program is working, I/O port 1 is used to indicate the program status. This information consists of which routine the program is in and under which mode it is operating. The main program sections are: Main loop, HSI interrupt, Timer 2 check, and Motor drive. There are also minor sections such as initialization, timer overflow handling, and software timer handling. Tying everything together is some overhead and glue. Where the glue is not obvious it will be discussed, otherwise it can be derived from the listings.

The program is a main loop which does nothing except serve as a place for the program to go when none of the interrupt routines are being run. All of the processing is done on an interrupt basis.

There are three basic software modes which are invoked depending on the speed of the motor. The modes referred to as 0, 1 and 2, in order from slowest to fastest operation. When the program is running the operating mode is indicated by the lower 2 bits of Port 1, with the following coding:

P1.0	P1.1	Mode	Description
0	0	0	HSI looks at every edge
1	0	1	HSI looks at Phase A edges only
0	1	2	Timer 2 used instead of HSI
1	1	2	(alternate form of above)

The example is easiest to see if mode 2 is described first, followed by mode 1 then mode 0. In mode 2 Timer 2 is used to count edges on the incoming signal. A software timer routine, which is actually run using HSO.0, uses the Timer 2 value to update a LONG (32-bit) software counter labeled *POSITION*. The HSO routine runs every 260 microseconds. The HSO.0 interrupt is used instead of an actual software timer because of the ability to easily unmask it while other software timer routines are running.

In the code in Listing 4-7, the mode is first determined. For the first pass ignore the code starting with the label *in_mode_1*. Starting with *in_mode_2* the counter is incremented or decremented based on bit zero of DIRECT. If DIRECT.0 = 0 the motor is going backward, if it is a 1 the motor is going forward. Next the count difference is checked to see if it is slow enough to go into mode 1. If not the routine returns to the code it was running when the interrupt occurred.

```
;;;
```

```
in_mode2:
        sub     delta_p,timer_2,tmr2_old        ; get timer2 count difference
        ld      tmr2_old,timer_2

        jbc     direct,0,in_rev

in_fwd: add     position,delta_p
        addc    position+2,zero
        br      chk_mode

in_rev: sub     position,delta_p
        subc    position+2,zero

chk_mode:
        sub     tmp1,Timer_2,old_t2     ; Check count difference in tmp1
        cmp     tmp1,#5                 ; set model if count is too low
        jgt     end_swt0                ; count <= 5

set_model:
        andb    Port1,#11111101B        ; Clear P1.1, set P1.0 (set mode 1)
        orb     Port1,#00000001B
        ldb     IOC0,#00000101B         ; enable HSI 0 and 1
        ld      zero, HSI_TIME
        sub     last1_time,Timer1,min_hsi1
                        ; set up so (time-last2_time)>min_hsi1 on next HSI
clr_hsi:
        ld      ZERO, HSI_TIME
        andb    iosl_bak,#01111111B             ; clear bit 7
        orb     iosl_bak,iosl
        jbs     iosl_bak,7,clr_hsi      ; If hsi is triggered then clear hsi

end_swt0:
        ld      old_t2,TIMER_2
        andb    port1,#11011111B        ; clear P1.5
        POPF
        ret
```

270061-51

Listing 4-7. Motor Control HSO.0 Timer Routine (Continued)

If the pulse rate is slow enough to go to mode 1, the transition is made by enabling HSI.0 and HSI.1. Both of these lines are connected to the same encoder line, with HSI.0 looking for rising edges and HSI.1 looking for falling edges. The *HSI_TIME* register is read to speed up clearing the HSI FIFO and the *LAST1_TIME* value is set up so the mode 1 routine does not immediately put the program into another mode. The HSI FIFO is then cleared, the Timer 2 value used throughout this routine is saved, and the routine returns.

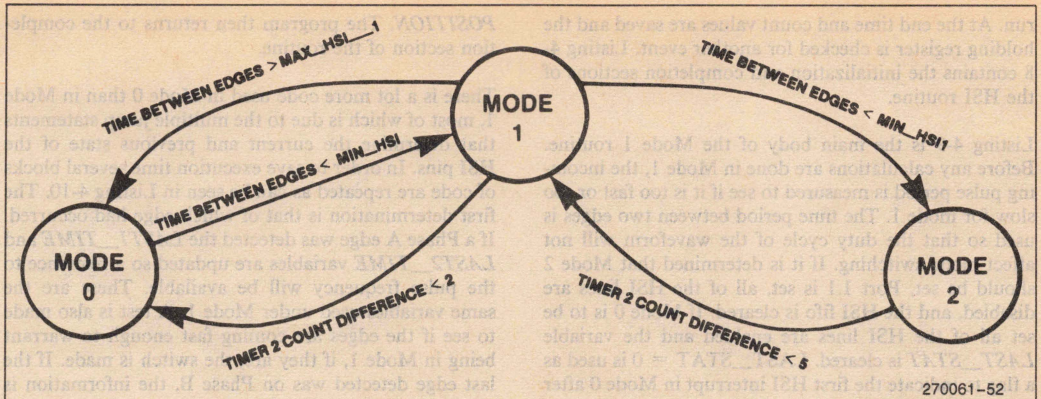
This routine still runs in modes 0 and 1, but in an abbreviated form. The section of code starting with the label *in_mode1* checks to see if the pulses are coming in so slowly that both HSI lines can be checked. If this is the case then all of the HSIs are enabled and the program returns. This routine is the secondary method for going from mode 1 to mode 0, the primary method is by checking the time between edges during the HSI routine, which will be described later.

The HSO routine will enable mode 0 from mode 1 if two edges are not received every 260 microseconds. The primary method, (under the HSI routine), can only enable mode 0 after an edge is received. This could cause a problem if the last 2 edges on Phase A before the encoder stops were too close to enable mode 0. If this happened, mode 0 would not be enabled until after the encoder started again, resulting in missed edges on Phase B. Using the HSO routine to switch from mode 1 to mode 0 eliminates this problem.

Figure 4-6 shows a state diagram of how the mode switching is done. As can be seen, there are two sources for most of the mode decisions. This helps avoid problems such as the one mentioned above.

When either Mode 1 or Mode 0 is enabled the HSI interrupt routine performs the counting of edges, while the HSO routine only ensures that the correct mode is running. The routines for modes 0 and 1 share the same initialization and completion sections, with the main body of code being different.

The initialization routine is similar to many HSI routines. The flags are checked to ensure that the HSI FIFO data is valid, and then the FIFO is read. Next, the main body of code (for either mode 0 or mode 1) is



NOTES:
Mode 0: HSI Examines edges on Phase A and B
Mode 1: HSI Examines edges on Phase A only
Mode 2: TIMER 2 stores edgecount

Figure 4-6. Mode State Diagram

```
;;;
```

run. At the end time and count values are saved and the holding register is checked for another event. Listing 4-8 contains the initialization and completion sections of the HSI routine.

Listing 4-9 is the main body of the Mode 1 routine. Before any calculations are done in Mode 1, the incoming pulse period is measured to see if it is too fast or too slow for mode 1. The time period between two edges is used so that the duty cycle of the waveform will not affect mode switching. If it is determined that Mode 2 should be set, Port 1.1 is set, all of the HSI lines are disabled, and the HSI fifo is cleared. If Mode 0 is to be set all of the HSI lines are enabled and the variable *LAST_STAT* is cleared. LAST_STAT = 0 is used as a flag to indicate the first HSI interrupt in Mode 0 after Mode 1. After the mode checking and setting are complete the incremental value in Timer 2 is used to update *POSITION*. The program then returns to the completion section of the routine.

There is a lot more code used in Mode 0 than in Mode 1, most of which is due to the multiple jump statements that determine the current and previous state of the HSI pins. In order to save execution time several blocks of code are repeated as can be seen in Listing 4-10. The first determination is that of which edge had occurred. If a Phase A edge was detected the *LAST1_TIME* and *LAST2_TIME* variables are updated so a reference to the pulse frequency will be available. These are the same variables used under Mode 1. A test is also made to see if the edges are coming fast enough to warrant being in Mode 1, if they are, the switch is made. If the last edge detected was on Phase B, the information is used only to determine direction.

```
In_mode_1:                  ; mode 1 HSI routine

        andb    tmp1,hsi_s0,#01010000B
        jne     no_cnt
cmp_time:                                   ; Procedure which sets mode 1 also
                                            ; sets times to pass the tests
        ld      last2_time,last1_time
        ld      last1_time,time

cmp1:   sub     tmp1,time,last2_time
        cmp     tmp1,min_hsi1
        jh      check_max_time

set_mode_2:
        orb     Port1,#00000010B            ; Set P1.1 (in mode 2)
        ldb     IOC0,#00000000B             ; Disable all HSI
mt_hsi: ld      zero,hsi_time               ; empty the hsi fifo
        andb    iosl_bak,#01111111B                 ; clear bit 7
        orb     iosl_bak,iosl
        jbs     iosl_bak,7,mt_hsi           ; If hsi is triggered then clear hsi
        br      done_chk

check_max_time:
        sub     tmp1,time,last2_time
        cmp     tmp1,max_hsi1               ; max_hsi = addition to min_hsi for
                                            ; total time
        jnh     done_chk

set_mode_0:
        andb    Port1,#11111100B            ; clear P1.0,1  set mode 0)
        ldb     IOC0,#01010101B             ; Enable all HSI
        ldb     last_stat,zero

done_chk:
        sub     delta_p,timer_2,tmr2_old            ; get timer2 count difference
        jbc     direct,0,add_rev
add_fwd:
        add     position,delta_p
        addc    position+2,zero
        br      load_lasts
add_rev:
        sub     position,delta_p
        subc    position+2,zero
        br      load_lasts

$eject
```

270061-54

Listing 4-9. Motor Control Mode 1 Routines

```
In_mode_0:
          jbs     hsi_s0,0,a_rise
          jbs     hsi_s0,2,a_fall
          jbs     hsi_s0,4,b_rise
          jbs     hsi_s0,6,b_fall
          br      no_cnt

a_rise:   ld      last2_time,last1_time
          ld      last1_time,time
          sub     time,last2_time
          cmp     time,min_hsi
          jh      tst_statr
;set model-
          orb     Port1,#00000001B          ; Set P1.0 (in mode 1)
          ldb     IOC0,#00000101B           ; Enable HSI 0 and 1
tst_statr:
          jbs     last_stat,6,going_fwd
          jbs     last_stat,4,going_rev
          jbs     last_stat,2,change_dir
          cmpb    last_stat,zero
          je      first_time                ; first time in mode0
          br      inp_err

a_fall:   ld      last2_time,last1_time
          ld      last1_time,time
          sub     time,last2_time
          cmp     time,min_hsi
          jh      tst_statf
;set model-
          orb     Port1,#00000001B          ; Set P1.0 (in mode 1)
          ldb     IOC0,#00000101B           ; Enable HSI 0 and 1
tst_statf:
          jbs     last_stat,4,going_fwd
          jbs     last_stat,6,going_rev
          jbs     last_stat,0,change_dir
          cmpb    last_stat,zero
          je      first_time                ; first time in mode0
          br      inp_err

b_rise:   jbs     last_stat,0,going_fwd
          jbs     last_stat,2,going_rev
          jbs     last_stat,6,change_dir
          cmpb    last_stat,zero
          je      first_time                ; first time in mode0
          br      inp_err

b_fall:   jbs     last_stat,2,going_fwd
          jbs     last_stat,0,going_rev
          jbs     last_stat,4,change_dir
          cmpb    last_stat,zero
          je      first_time                ; first time in mode0
          br      inp_err

first_time:
          stb     hsi_s0,last_stat
          br      done_chk          ; add delta position
inp_err:
          br      no_int


change_dir:
          notb    direct
no_inc:   jbc     direct,0,going_rev

going_fwd:
          orb     PORT2,#01000000B          ; set P2.6
          ldb     direct,#01                ; direction = forward
          add     position,#01
          addc    position+2,zero
          br      st_stat
going_rev:
          andb    PORT2,#10111111B          ; clear P2.6
          ldb     direct,#00                ; direction = reverse
          sub     position,#01
          subc    position+2,zero

st_stat:
          stb     hsi_s0,last_stat
```

270061-55

Listing 4-10. Motor Control Mode 0 Routines

After mode correctness is confirmed and the *LASTx__TIME* values are updated the *LAST__STAT* (Last Status) variable is used to determine the current direction of travel. The POSITION value is then updated in the direction specified by the last two edges and the status is stored. Note that the first time in Mode 0 after being in Mode 1, the Mode 1 *done__chk* routine is used to update POSITION, instead of the routines *going__fwd* and *going__rev* from the Mode 0 section of code. The completion section of code is then executed.

Providing the PWM value to drive the motor is done by a routine running under Software Timer 1. The first section of code, shown in Listing 4-11a, has to do with calculating the position and timer errors. Listing 4-11b shows the next section of code where the power to be supplied to the motor is calculated. First the direction is checked and if the direction is reverse the absolute value of the error is taken. If the error is greater than 64K counts, the PWM routine is loaded with the maximum value. The next check is made to see if the motor is close enough to the desired location that the power to it should be reversed, (i.e., enter the Braking mode). If the motor is very close to the position or has slowed to the point that is likely to turn around, the *Hold__Position mode is entered.*

The determination of which modes are selected under what conditions was done empirically. All of the parameters used to determine the mode are kept in RAM so they can be easily changed on the fly instead of by re-assembling the program. The parameters in the listing have been selected to make the motor run, but have not been optimized for speed or stability. A diagram of the modes is shown in Figure 4-7.

In the *Hold__Position* mode power is eased onto the motor to lock it into position. Since the motor could be stopped in this mode, some integral control is needed, as proportional control alone does not work well when the error is small and the load is large. The BOOST variable provides this integral control by increasing the output a fixed amount every time period in which the

Listing 4-11. Motor Control Software Timer 1 Routine

```
;;;
```

```
chk_dir:
        cmp     pos_err+2,zero
        jge     go_forward

go_backward:
        neg     pos_err             ; Pos_err = ABS VAL (pos_err)
        ldb     pwm_dir,#00h
        cmp     pos_err+2,#0ffffH
        jne     ld_max
        br      chk_brk

go_forward:
        ldb     pwm_dir,#01H
        cmp     pos_err+2,zero
        je      chk_brk

ld_max: ldb     pwm_pwr,max_pwr
        br      chk_sanity

Chk_brk:                            ;  Position_Error now = ABS(pos_err)
        cmp     pos_err,pos_pnt
        jnh     hold_position       ; position_error<position_control_point
        cmp     pos_err,brk_pnt
        jh      ld_max              ; position_error>brake_point

braking:
        cmp     pos_delta,zero
        jge     chk_delta
        neg     pos_delta
chk_delta:
        cmp     pos_delta,vel_pnt           ; velocity = pos_delta/sample_time
        jnh     hold_position               ; jmp if  ABS(velocity) < vel_pnt

brake:  ldb     pwm_pwr,max_brk
        ldb     tmp,direct                  ; If braking apply power in opposite
        notb    tmp                         ; direction of current motion
        ldb     pwm_dir,tmp

        br      ld_pwr

Hold_position:                      ; position hold mode
        cmp     pos_err,#02
        jh      calc_out            ; if position error < 2 then turn off power
        clr     tmp+2
        clr     boost
        BR      output

calc_out:
        mulub   tmp,max_hold,#255
        mulu    tmp,pos_err                 ; Tmp = pos_err * max_hold
        cmp     pos_delta,zero
        jne     no_bst
        add     boost,#04                   ; Boost is integral control
        add     tmp+2,boost                 ; TMP+2 = MSB(pos_err*max_hold)
        br      ck_max
no_bst: clr     boost
ck_max: cmp     tmp+2,max_hold
        jnh     output
maxed:  ld      tmp+2,max_hold
output: ldb     pwm_pwr,tmp+2

chk_sanity:
        br      ld_pwr

ld_pwr:
        ldb     rpwr,pwm_pwr
        notb    rpwr
        jbs     pwm_dir,0,p2fwd

p2bkwd: DI
        andb    port2,#01111111B            ; clear P2.7
        ldb     pwm_control,rpwr
        EI
        br      pwrset
p2fwd:  DI
        orb     port2,#10000000B            ; set P2.7
        ldb     pwm_control,rpwr
        EI
```

270061-57

Listing 4-11b. Motor Control Power Algorithm

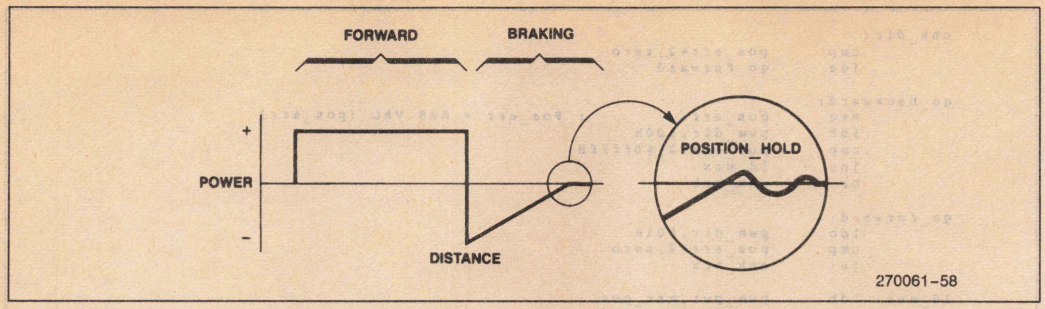


Figure 4-7. Motor Control Modes

error does not get smaller. Once the error does get smaller, usually because the motor starts moving, BOOST is cleared.

A sanity check can be performed at this point to double check that the 8096 has proper control of the motor. In the example the worst that can happen is the proto-

```
pwrset:
        cmp     time_err+2,zero  ; do pos_table when err is negative
        jgt     end_p
;;;     br      end_p

        cmp     nxt_pos,#(32+pos_table)
        jlt     get_vals                ; jump if lower
        ld      nxt_pos,#pos_table
        clr     time+2
get_vals:

        ld      des_pos,[nxt_pos]+
        ld      des_pos+2,[nxt_pos]+
        ld      des_time+2,[nxt_pos]+
        ld      max_pwr,[nxt_pos]+
        ld      max_brk,max_pwr
        add     des_pos,offset
        addc    des_pos+2,zero
        sub     last_pos_err,des_pos,position

end_p:  andb    port1,#01111111B        ; clear P1.7

        popf
        ret

pos_table:

        dcl     00000000H       ; position 0
        dcw     0020H, 0080H    ; next time, power
        dcl     0000c000H       ; position 1
        dcw     0040H, 0040H    ; next time, power
        dcl     00000000H       ; position 2
        dcw     0060H, 00c0H    ; next time, power
        dcl     0FFFF8000H      ; position 3
        dcw     0080H, 0080H    ; next time, power

        dcl     00000800H       ; position 4
        dcw     0058H, 0080H    ; next time, power
        dcl     00003000H       ; position 5
        dcw     0070H, 00ffH    ; next time, power
        dcl     00000000H       ; position 6
        dcw     0090H, 00f0H    ; next time, power
        dcl     00000000H       ; position 7
        dcw     0091H, 00f0H    ; next time, power
```

270061-59

Listing 4-12. Motor Control Next Position Lookup

type will need to be reset, so the sanity check was not used. If one were desired, it could be as simple as checking a hardware generated direction indicator, or as complex as checking motor condition and other environmental factors.

After all checks have been made, the power value is loaded to the RPWR register using a software inversion to compensate for the hardware inversion. Direction is determined next and the power and direction are changed in adjacent instructions with interrupts disabled to prevent changing power without direction and vice versa.

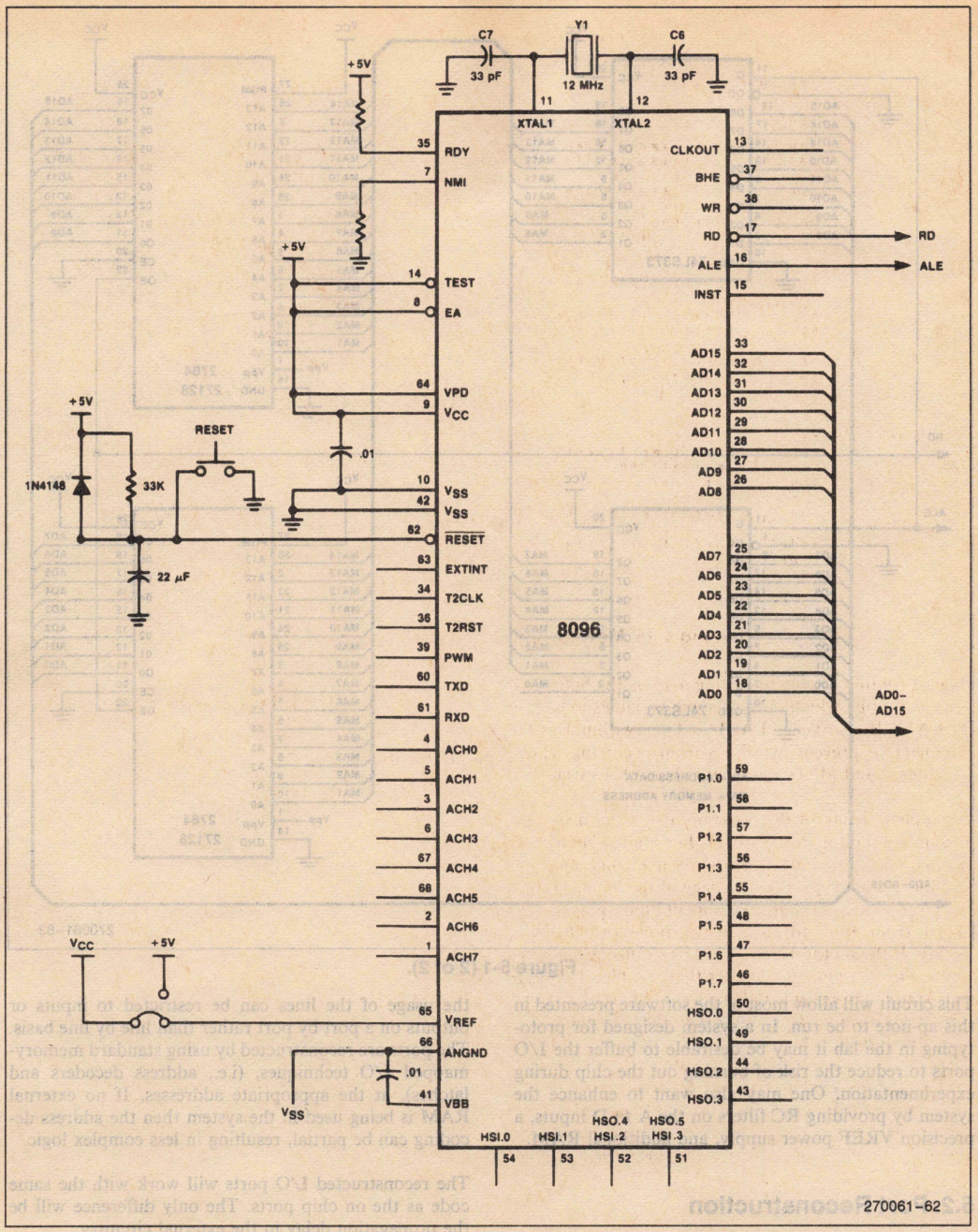
To exercise the program logic the desired position is changed based on the time value using the code and lookup table shown in Listing 4-12.

The remaining sections of the program are relatively simple, but worth discussing briefly. The initialization routine initializes the I/O features and places several variables from ROM into RAM. Having these variables in RAM makes it easier to tweak the algorithm. Timer 1 is expanded into a 32-bit timer by the interrupt routine shown in Listing 4-13.

Software timer overhead is handled by the routine shown in Listing 4-14. In this routine the status of each timer bit is checked in a shadow register. If any of the timers have expired the appropriate routine is called.

```
;;;
```

```
;;;
```



one contains the odd bytes, and the addressing is not fully decoded. This means that the addressing on a 2764 will be such that the lower 4K of each EPROM is mapped at 0000H and 4000H while the upper 4K is mapped at 2000H. If the program being loaded is 16 Kbytes long the first half is loaded into the second half of the 2764s and vice versa. A similar situation exists when using 27128s.

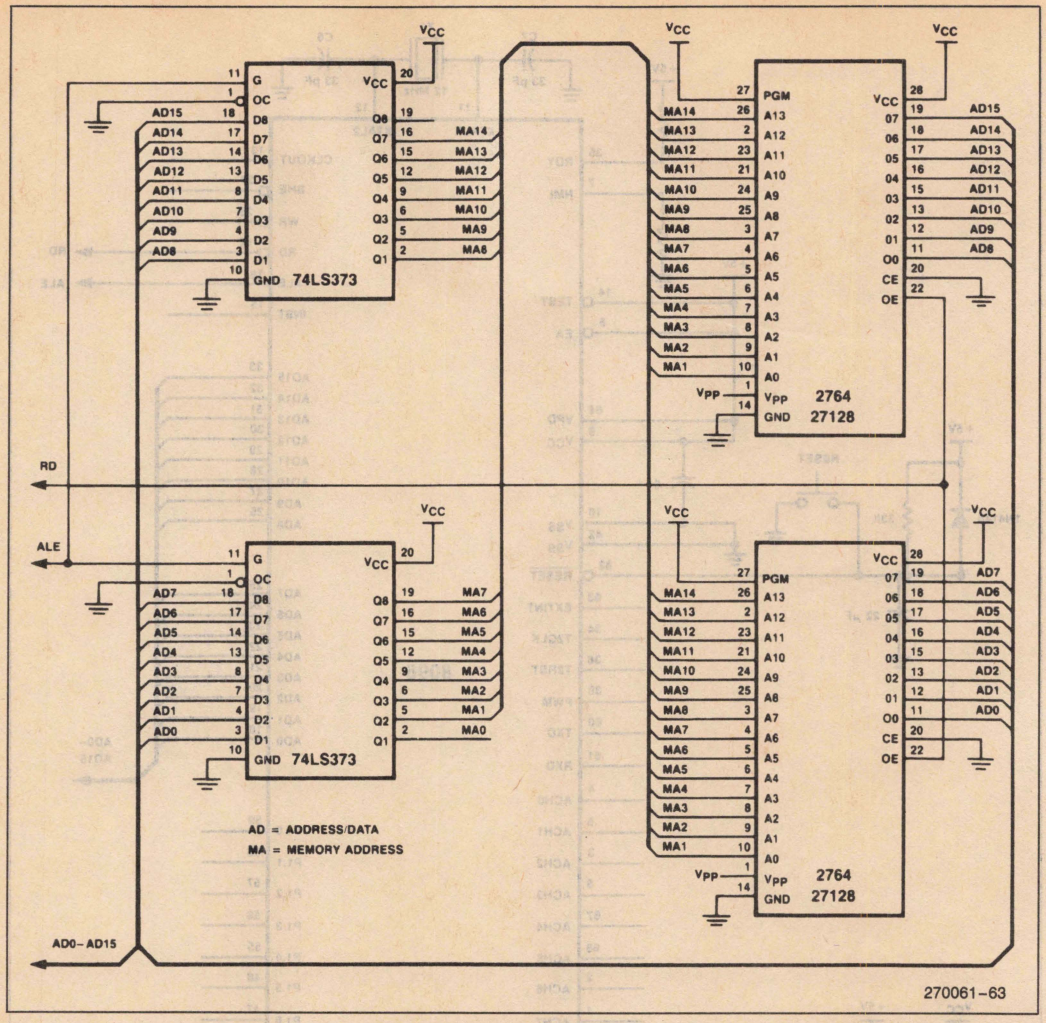


Figure 5-1 (2 of 2).

This circuit will allow most of the software presented in this ap-note to be run. In a system designed for prototyping in the lab it may be desirable to buffer the I/O ports to reduce the risk of burning out the chip during experimentation. One may also want to enhance the system by providing RC filters on the A to D inputs, a precision VREF power supply, and additional RAM.

5.2. Port Reconstruction

If it is desired to fully emulate a 8396 then I/O ports 3 and 4 must be reconstructed. It is easiest to do this if the usage of the lines can be restricted to inputs or outputs on a port by port rather than line by line basis. The ports are reconstructed by using standard memory-mapped I/O techniques, (i.e., address decoders and latches), at the appropriate addresses. If no external RAM is being used in the system then the address decoding can be partial, resulting in less complex logic.

The reconstructed I/O ports will work with the same code as the on chip ports. The only difference will be the propagation delay in the external circuitry.

6.0 CONCLUSION

An overview of the MCS-96 family has been presented along with several simple examples and a few more complex ones. The source code for all of these programs are available in the Insite Users Library using order code AE-16. Additional information on the 8096 can be found in the Microcontroller Handbook and it is recommended that this book be in your possession before attempting any work with the MCS-96 family of products. Your local Intel sales office can assist you in getting more information on the 8096 and its hardware and software development tools.

7.0 BIBLOGRAPHY

1. MSC-96 Macro Assembler User's Guide, Intel Corporation, 1983.

   Order number 122048-001.

2. Microcontroller Handbook (1985), Intel Corporation, 1984.

   Order number 210918-002.

3. MSC-96 Utilities User's Guide, Intel Corporation, 1983.

   Order number 122049-001.

4. PL/M-96 User's Guide, Intel Corporation, 1983.

   Order number 122134-001.

```
SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE:  F3:INTER1.A96
OBJECT FILE:  F3:INTER1.OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND:  NOSB

ERR LOC  OBJECT                  LINE      SOURCE STATEMENT
                                    1   $TITLE('INTER1 A96:  Interpolation routine 1')
                                    2   ;;;;;;;   8096 Assembly code for table lookup and interpolation
                                    3
                                    4   $INCLUDE(:F0:DEMO96.INC)      ; Include demo definitions
                                =1  5   $nolist  ;  Turn listing off for include file
                                =1 53           ;  End of include file
                                   54
    0022                           55   RSEG  at 22H
                                   56
    0022                           57           IN_VAL:          dsb     1            ; Actual Input Value
    0024                           58           TABLE_LOW:       dsw     1
    0026                           59           TABLE_HIGH:      dsw     1
    0028                           60           IN_DIF:          dsw     1            ; Upper Input - Lower Input
      0028                         61           IN_DIFB          equ     IN_DIF  :byte
    002A                           62           TAB_DIF:         dsw     1            ; Upper Output - Lower Output
    002C                           63           OUT:             dsw     1
    002E                           64           RESULT:          dsw     1
    0030                           65           OUT_DIF:         dsl     1            ; Delta Out
                                   66
                                   67
    2080                           68   CSEG at 2080H
                                   69
    2080 A1000118                  70           LD      SP, #100H
                                   71
    2084 B0221C                    72   look:   LDB     AL, IN_VAL      ; Load temp with Actual Value
    2087 18031C                    73           SHRB    AL, #3          ; Divide the byte by 8
    208A 71FE1C                    74           ANDB    AL, #11111110B ; Insure AL is a word address
                                   75                                   ;  This effectively divides AL by 2
                                   76                                   ;  so AL = IN_VAL/16
                                   77
    208D AC1C1C                    78           LDBZE   AX, AL          ; Load byte AL to word AX
    2090 A31D002124                79           LD      TABLE_LOW, TABLE [AX]    ; TABLE_LOW is loaded with the value
                                   80                                            ;  in the table at table location AX
                                   81
```

270061-64

```
2095 A31D022126          82          LD      TABLE_HIGH, (TABLE+2)[AX] ; TABLE_HIGH is loaded with the
                         83                                            ; value in the table at table
                         84                                            ; location AX+2
                         85                                            ; (The next value in the table)
                         86
209A 4824262A            87          SUB     TAB_DIF, TABLE_HIGH, TABLE_LOW
                         88                                            ; TAB_DIF=TABLE_HIGH-TABLE_LOW
                         89
209E 510F2228            90          ANDB    IN_DIFB, IN_VAL, #0FH     ; IN_DIFB=least significant 4 bits
                         91                                            ;  of IN_VAL
20A2 AC2828              92          LDBZE   IN_DIF, IN_DIFB           ; Load byte IN_DIFB to word IN_DIF
                         93
20A5 FE4C2A2830          94          MUL     OUT_DIF, IN_DIF, TAB_DIF
                         95                                            ; Output_difference =
                         96                                            ;  Input_difference*Table_difference
20AA 0E0430              97          SHRAL   OUT_DIF, #4               ; Divide by 16 (2**4)
                         98
20AD 4424302C            99          ADD     OUT, OUT_DIF, TABLE_LOW ; Add output difference to output
                        100                                            ;  generated with truncated IN_VAL
                        101                                            ;  as input
20B1 0A042C             102          SHRA    OUT, #4                   ; Round to 12-bit answer
20B4 A4002C             103          ADDC    OUT, zero                 ; Round up if Carry = 1
                        104
20B7 C02E2C             105  no_inc: ST      OUT, RESULT               ; Store OUT to RESULT
                        106
20BA 27C8               107          BR      look                      ; Branch to "look:"
                        108
                        109
2100                    110  cseg    AT 2100H
                        111
2100 0000002000034004C  112  table:  DCW     0000H, 2000H, 3400H, 4C00H  ; A random function
2108 005D006A00720078   113          DCW     5D00H, 6A00H, 7200H, 7800H
2110 007B007D0076006D   114          DCW     7B00H, 7D00H, 7600H, 6D00H
2118 005D004B00340022   115          DCW     5D00H, 4B00H, 3400H, 2200H
2120 0010               116          DCW     1000H
                        117
2122                    118  END

ASSEMBLY COMPLETED,   NO ERROR(S) FOUND.
```

270061-65

```
SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE:   F3:INTER2.A96
OBJECT FILE:   F3:INTER2.OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND:  NOSB

ERR LOC  OBJECT                  LINE        SOURCE STATEMENT
                                    1   $TITLE('INTER2 A96: Interpolation routine 2')
                                    2
                                    3   ;;;;;;;   8096 Assembly code for table lookup and interpolation
                                    4   ;;;;;;;   Using tabled values in place of division
                                    5
                                    6   $INCLUDE(:F0:DEMO96.INC) ; Include demo definitions
                               =1   7   $nolist  ;  Turn listing off for include file
                               =1  55            ;  End of include file
                                   56
    0024                           57   RSEG  at 24H
                                   58
    0024                           59          IN_VAL:         dsb     1           ; Actual Input Value
    0026                           60          TABLE_LOW:      dsw     1           ; Table value for function
    0028                           61          TABLE_INC:      dsw     1           ; Incremental change in function
    002A                           62          IN_DIF:         dsw     1           ; Upper Input - Lower Input
      002A                         63          IN_DIFB         equ     IN_DIF  :byte
    002C                           64          OUT:            dsw     1
    002E                           65          RESULT:         dsw     1
    0030                           66          OUT_DIF:        dsl     1           ; Delta Out
                                   67
                                   68
    2080                           69   CSEG at 2080H
                                   70
    2080 A1000118                  71          LD      SP, #100H       ; Initialize SP to top of reg  file
                                   72
    2084 B0241C                    73   look:  LDB     AL, IN_VAL      ; Load temp with Actual Value
    2087 18031C                    74          SHRB    AL, #3          ; Divide the byte by 8
    208A 71FE1C                    75          ANDB    AL, #11111110B  ; Insure AL is a word address
                                   76                                  ;  This effectively divides AL by 2
                                   77                                  ;  so AL = IN_VAL/16
    208D AC1C1C                    78          LDBZE   AX, AL          ; Load byte AL to word AX
                                   79
    2090 A31D002126                80          LD      TABLE_LOW, VAL_TABLE[AX] ; TABLE_LOW is loaded with the value
                                   81                                           ;  in the value table at location AX
                                   82
    2095 A31D222128                83          LD      TABLE_INC, INC_TABLE[AX] ; TABLE_INC is loaded with the value
                                   84                                           ; in the increment table at
                                   85                                           ; location AX+2
                                   86
```

270061-66

```
209A 510F242A          87              ANDB    IN_DIFB, IN_VAL, #0FH   ; IN_DIFB=least significant 4 bits
                       88                                              ;  of IN_VAL
209E AC2A2A            89              LDBZE   IN_DIF, IN_DIFB         ; Load byte IN_DIFB to word IN_DIF
                       90
20A1 FE4C282A30        91              MUL     OUT_DIF, IN_DIF, TABLE_INC
                       92                                              ; Output_difference =
                       93                                              ;  Input_difference*Incremental_change
                       94
20A6 4426302C          95              ADD     OUT, OUT_DIF, TABLE_LOW ; Add output difference to output
                       96                                              ;  generated with truncated IN_VAL
                       97                                              ;  as input
20AA 08042C            98              SHR     OUT, #4                 ; Round to 12-bit answer
20AD A4002C            99              ADDC    OUT, zero               ; Round up if Carry = 1
                      100
20B0 C02E2C           101   no_inc:   ST      OUT, RESULT             ; Store OUT to RESULT
20B3 27CF             102             BR      look                    ; Branch to "look:"
                      103
                      104
2100                  105   cseg      AT 2100H
                      106
2100                  107   val_table:
2100 000000200034004C 108             DCW     0000H, 2000H, 3400H, 4C00H  ; A random function
2108 005D006A00720078 109             DCW     5D00H, 6A00H, 7200H, 7800H
2110 007B007D0076006D 110             DCW     7B00H, 7D00H, 7600H, 6D00H
2118 005D004B00340022 111             DCW     5D00H, 4B00H, 3400H, 2200H
2120 0010             112             DCW     1000H
2122                  113   inc_table:
2122 0002400180011001 114             DCW      0200H,  0140H,  0180H,  0110H   ; Table of incremental
212A D000800060003000 115             DCW      00D0H,  0080H,  0060H,  0030H   ; differences
2132 200090FF70FF00FF 116             DCW     0020H, 0FF90H, 0FF70H, 0FF00H
213A E0FE90FEE0FEE0FE 117             DCW     0FEE0H, 0FE90H, 0FEE0H, 0FEE0H
                      118
2142                  119   END

ASSEMBLY COMPLETED,   NO ERROR(S) FOUND.
```

270061-67

```
SERIES-III PL/M-96 V1.0 COMPILATION OF MODULE PLMEX
OBJECT MODULE PLACED IN :F3:PLMEX1.OBJ
COMPILER INVOKED BY:  PLM96.86 :F3:PLMEX1.P96 CODE


                $TITLE('PLMEX1:  PLM-96 Example Code for Table Lookup')

                /* PLM-96 CODE FOR TABLE LOOK-UP AND INTERPOLATION */

  1             PLMEX:    DO;

  2     1       DECLARE IN_VAL          WORD        PUBLIC;
  3     1       DECLARE TABLE_LOW       INTEGER     PUBLIC;
  4     1       DECLARE TABLE_HIGH      INTEGER     PUBLIC;
  5     1       DECLARE TABLE_DIF       INTEGER     PUBLIC;
  6     1       DECLARE OUT             INTEGER     PUBLIC;
  7     1       DECLARE RESULT          INTEGER     PUBLIC;
  8     1       DECLARE OUT_DIF         LONGINT     PUBLIC;
  9     1       DECLARE TEMP            WORD        PUBLIC;

 10     1       DECLARE TABLE(17)       INTEGER  DATA (
                        0000H, 2000H, 3400H, 4C00H,       /* A random function */
                        5D00H, 6A00H, 7200H, 7800H,
                        7B00H, 7D00H, 7600H, 6D00H,
                        5D00H, 4B00H, 3400H, 2200H,
                        1000H);

 11     1       DMPY:   PROCEDURE (A,B) LONGINT EXTERNAL;
 12     2               DECLARE (A,B) INTEGER;
 13     2       END DMPY;

 14     1       LOOP
                    TEMP=SHR(IN_VAL,4);     /* TEMP is the most significant 4 bits of IN_VAL */

 15     1          TABLE_LOW=TABLE(TEMP);      /* If "TEMP" was replaced by "SHR(IN_VAL,4)"  */
 16     1          TABLE_HIGH=TABLE(TEMP+1);   /* The code would work but the 8096 would     */
                                                /* do two shifts                              */

 17     1          TABLE_DIF=TABLE_HIGH-TABLE_LOW;

 18     1          OUT_DIF=DMPY(TABLE_DIF,SIGNED(IN_VAL AND 0FH)) /16;

 19     1          OUT=SAR((TABLE_LOW+OUT_DIF),4); /* SAR performs an arithmetic right shift,
                                                      in this case 4 places are shifted     */
```

270061-68

```
20   1          IF CARRY=0 THEN RESULT=OUT;   /* Using the hardware flags must be done   */
22   1            ELSE  RESULT=OUT+1;         /* with care to ensure the flag is tested  */
                                               /* in the desired instruction sequence     */
23   1      GOTO LOOP;

           /* END OF PLM-96 CODE  */

24   1      END;
```

270061-69

```
PL/M-96 COMPILER     PLMEX1:  PLM-96 Example Code for Table Lookup
                     ASSEMBLY LISTING OF OBJECT CODE


                                        ;   STATEMENT     14
          0022                      PLMEX:
          0022  A1000018          R        LD    SP,#STACK
          0026                    LOOP:
          0026  A00010            R        LD    TEMP,IN_VAL
          0029  0B0410            R        SHR   TEMP,#4H
                                           ;   STATEMENT     15
          002C  4410101C          R        ADD   TMP0,TEMP,TEMP
          0030  A31D000002        R        LD    TABLE_LOW,TABLE[TMP0]
                                           ;   STATEMENT     16
          0035  A31D020004        R        LD    TABLE_HIGH,TABLE+2H[TMP0]
                                           ;   STATEMENT     17
          003A  48020406          R        SUB   TABLE_DIF,TABLE_HIGH,TABLE_LOW
                                           ;   STATEMENT     18
          003E  C806              R        PUSH  TABLE_DIF
          0040  410F00001C        R        AND   TMP0,IN_VAL,#0FH
          0045  C81C                       PUSH  TMP0
          0047  EF0000            E        CALL  DMPY
          004A  0E041C                     SHRAL TMP0,#4H
          004D  A01E0E            R        LD    OUT_DIF+2H,TMP2
          0050  A01C0C            R        LD    OUT_DIF,TMP0
                                           ;   STATEMENT     19
          0053  A00220            R        LD    TMP4,TABLE_LOW
          0056  0620                       EXT   TMP4
          0058  641C20                     ADD   TMP4,TMP0
          005B  A41E22                     ADDC  TMP6,TMP2
          005E  0E0420                     SHRAL TMP4,#4H
          0061  A02008            R        LD    OUT,TMP4
                                           ;   STATEMENT     20
          0064  B1FF1C                     LDB   TMP0,#0FFH
          0067  DB02                       BC    @0003
          0069  111C                       CLRB  TMP0
          006B                    @0003:
```

270061-70

```
006B  981C00                        CMPB   R0,TMP0
006E  D705                          BNE    @0001
                                    ;    STATEMENT     21
0070  A0200A              R         LD     RESULT,TMP4
0073  2005                         BR     @0002
                                    ;    STATEMENT     22
0075                          @0001:
0075  A0080A              R         LD     RESULT,OUT
0078  070A                R         INC    RESULT
                                    ;    STATEMENT     23
007A                          @0002:
007A  27AA                          BR     LOOP
                                    ;    STATEMENT     24
                                    END


MODULE INFORMATION:

     CODE AREA SIZE                = 005AH     90D
     CONSTANT AREA SIZE            = 0022H     34D
     DATA AREA SIZE                = 0000H      0D
     STATIC REGS AREA SIZE         = 0012H     18D


PL/M-96 COMPILER     PLMEX1   PLM-96 Example Code for Table Lookup
                     ASSEMBLY LISTING OF OBJECT CODE


     OVERLAYABLE REGS AREA SIZE = 0000H      0D
     MAXIMUM STACK SIZE         = 0006H      6D
     48 LINES READ

PL/M-96 COMPILATION COMPLETE.       0 WARNINGS,     0 ERRORS
```

270061-71

```
MCS-96 MACRO ASSEMBLER    MULT.APT:  16*16 multiply procedure for PLM-96

SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE:  :F3:MULT.A96
OBJECT FILE:  :F3:MULT.OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND:  NOSB

ERR LOC  OBJECT                  LINE        SOURCE STATEMENT
                                    1   $TITLE('MULT.APT:  16*16 multiply procedure for PLM-96')
                                    2
                                    3
      0018                          4           SP      EQU     18H:word
                                    5
     0000                           6   rseg
                                    7           EXTRN   PLMREG  :long
                                    8
     0000                           9   cseg
                                   10
                                   11           PUBLIC  DMPY          ; Multiply two integers and return a
                                   12                                 ; longint result in AX, DX registers
                                   13
     0000 CC04              E     14   DMPY:   POP     PLMREG+4              ; Load return address
     0002 CC00              E     15           POP     PLMREG                ; Load one operand
     0004 FE6E1900          E     16           MUL     PLMREG,[SP]+    ; Load second operand and increment SP
                                   17
     0008 E304              E     18           BR      [PLMREG+4]          ; Return to PLM code.
     000A                         19   END

ASSEMBLY COMPLETED,  NO ERROR(S) FOUND.
```

270061-72

```
SERIES-III MCS-96 RELOCATOR AND LINKER, V2.0
Copyright 1983 Intel Corporation

INPUT FILES:  :F3:PLMEX1.OBJ, :F3:MULT.OBJ, PLM96.LIB
OUTPUT FILE:  :F3:PLMOUT.OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND:
   ROM(2080H-3FFFH)


INPUT MODULES INCLUDED:
   :F3:PLMEX1.OBJ(PLMEX)  12/25/84
   :F3:MULT.OBJ(MULT)  12/25/84
   PLM96.LIB(PLMREG)  11/02/83


SEGMENT MAP FOR :F3:PLMOUT.OBJ(PLMEX):

             TYPE    BASE      LENGTH    ALIGNMENT    MODULE NAME
             ----    ----      ------    ---------    -----------

**RESERVED*          0000H     001AH
*** GAP ***          001AH     0002H
             REG     001CH     0008H     ABSOLUTE     PLMREG
             REG     0024H     0012H     WORD         PLMEX
             STACK   0036H     0006H     WORD
*** GAP ***          003CH     2044H
             CODE    2080H     0003H     ABSOLUTE     PLMEX
*** GAP ***          2083H     0001H
             CODE    2084H     007CH     WORD         PLMEX
             CODE    2100H     000AH     BYTE         MULT
*** GAP ***          210AH     DEF6H
```

270061-73

```
SYMBOL TABLE FOR :F3:PLMOUT.OBJ(PLMEX).

ATTRIBUTES              VALUE     NAME
----------              -----     ----

                                  PUBLICS:
REG      WORD           0024H       IN_VAL
REG      INTEGER        0026H       TABLE_LOW
REG      INTEGER        0028H       TABLE_HIGH
REG      INTEGER        002AH       TABLE_DIF
REG      INTEGER        002CH       OUT
REG      INTEGER        002EH       RESULT
REG      LONGINT        0030H       OUT_DIF
REG      WORD           0034H       TEMP
CODE     ENTRY          2100H       DMPY
REG      LONG           001CH       PLMREG
NULL     NULL           003CH       MEMORY
NULL     NULL           1FC4H       ?MEMORY_SIZE

                                  MODULE: PLMEX

                                  MODULE: MULT

                                  MODULE: PLMREG

RL96 COMPLETED,   0 WARNING(S),   0 ERROR(S)
```

270061-74

```
SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE:   F3:PULSE.A96
OBJECT FILE:   F3:PULSE.OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND:  NOSB

ERR LOC  OBJECT                LINE        SOURCE STATEMENT
                                  1   $TITLE('PULSE.A96: Measuring pulses using the HSI unit')
                                  2
                                  3   $INCLUDE(DEMO96.INC)
                              =1  4   $nolist  ;  Turn listing off for include file
                              =1 52            ;  End of include file
                                 53
    0028                         54   rseg     at  28H
                                 55
    0028                         56            HIGH_TIME:      dsw     1
    002A                         57            LOW_TIME:       dsw     1
    002C                         58            PERIOD:         dsw     1
    002E                         59            HI_EDGE:        dsw     1
    0030                         60            LO_EDGE:        dsw     1
                                 61
                                 62
                                 63
    2080                         64   cseg     at  2080H
                                 65
                                 66
    2080 A1000118                67            LD      SP, #100H
    2084 B10115                  68            LDB     IOCO, #00000001B        ; Enable HSI 0
    2087 B10F03                  69            LDB     HSI_MODE, #00001111B    ; HSI 0 look for either edge
                                 70
    208A 442A282C                71   wait:    ADD     PERIOD, HIGH_TIME, LOW_TIME
    208E 3E1603                  72            JBS     IOS1, 6, contin         ; If FIFO is full
    2091 3716F6                  73            JBC     IOS1, 7, wait   ; Wait while no pulse is entered
                                 74
    2094 B0061C                  75   contin:  LDB     AL, HSI_STATUS          ; Load status; Note that reading
                                 76                                           ;   HSI_TIME clears HSI_STATUS
                                 77
    2097 A00420                  78            LD      BX, HSI_TIME            ; Load the HSI_TIME
                                 79
    209A 391C09                  80            JBS     AL, 1, hsi_hi           ; Jump if HSI.0 is high
                                 81
    209D C03020                  82   hsi_lo:  ST      BX, LO_EDGE
    20A0 482E3028                83            SUB     HIGH_TIME, LO_EDGE, HI_EDGE
    20A4 27E4                    84            BR      wait
                                 85
                                 86
    20A6 C02E20                  87   hsi_hi:  ST      BX, HI_EDGE
```

270061-75

```
20A9 48302E2A                88          SUB     LOW_TIME, HI_EDGE, LO_EDGE
20AD 27DB                    89          BR      wait
                             90
20AF                         91          END

ASSEMBLY COMPLETED,  NO ERROR(S) FOUND.
```

270061-76

```
SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE    F3 ENHSI.A96
OBJECT FILE    F3 ENHSI.OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND:  NOSB

ERR LOC  OBJECT                  LINE        SOURCE STATEMENT
                                    1   $TITLE ('ENHSI A96  ENHANCED HSI PULSE ROUTINE')
                                    2
                                    3   $INCLUDE(DEMO96.INC)
                               =1   4   $nolist  ;  Turn listing off for include file
                               =1  52            ;  End of include file
                                   53
    0028                           54   RSEG AT 28H
                                   55
    0028                           56            TIME:           DSW 1
    002A                           57            LAST_RISE:      DSW 1
    002C                           58            LAST_FALL:      DSW 1
    002E                           59            HSI_SO:         DSB 1
    002F                           60            IOS1_BAK:       DSB 1
    0030                           61            PERIOD:         DSW 1
    0032                           62            LOW_TIME:       DSW 1
    0034                           63            HIGH_TIME:      DSW 1
    0036                           64            COUNT:          DSW 1
                                   65
    2080                           66   cseg     at      2080H
                                   67
    2080 A1000118                  68   init:    LD      SP,#100H
                                   69
    2084 B12516                    70            LDB     IOC1,#00100101B ; Disable HSO.4,HSO.5, HSI_INT=first,
                                   71                                    ; Enable PWM,TXD,TIMER1_OVRFLOW_INT
                                   72
    2087 B19903                    73            LDB     HSI_MODE,#10011001B     ; set hsi.1 -; hsi.0 +
    208A B10715                    74            LDB     IOC0,#00000111B         ; Enable  hsi 0,1
                                   75                                            ; T2 CLOCK=T2CLK,  T2RST=T2RST
                                   76                                            ; Clear timer2
                                   77
                                   78
    208D 717F2F                    79   wait:    ANDB    IOS1_BAK,#01111111B     ; Clear IOS1_BAK.7
    2090 90162F                    80            ORB     IOS1_BAK,IOS1           ; Store into temp to avoid clearing
                                   81                                            ; other flags which may be needed
    2093 372FF7                    82            JBC     IOS1_BAK,7,wait         ; If hsi is not triggered then
                                   83                                            ; jump to wait
                                   84
    2096 5155062E                  85            ANDB    HSI_SO,HSI_STATUS,#01010101B
    209A A00428                    86            LD      TIME, HSI_TIME
                                   87
```

270061-77

```
209D 382E05                88          JBS     HSI_SO,0,a_rise
20A0 3A2E0F                89          JBS     HSI_SO,2,a_fall
20A3 201A                  90          BR      no_cnt
                           91
20A5 482C2832              92  a_rise: SUB     LOW_TIME, TIME, LAST_FALL
20A9 482A2830              93          SUB     PERIOD, TIME,LAST_RISE
20AD A0282A                94          LD      LAST_RISE, TIME
20B0 200B                  95          BR      increment
                           96
20B2 482A2834              97  a_fall: SUB     HIGH_TIME, TIME, LAST_RISE
20B6 482C2830              98          SUB     PERIOD, TIME,LAST_FALL
20BA A0282C                99          LD      LAST_FALL, TIME
                           100
20BD                       101 increment:
20BD 0736                  102         INC     COUNT
20BF 27CC                  103 no_cnt: BR      wait
                           104
20C1                       105         END

ASSEMBLY COMPLETED,   NO ERROR(S) FOUND.
```

270061-78

```
SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE:  :F3:HSODRV.A96
OBJECT FILE:  :F3:HSODRV.OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND:  NOSB

ERR LOC  OBJECT                  LINE       SOURCE STATEMENT
                                    1   $TITLE('HSODRV.A96:  Driver module for HSO PWM program')
                                    2
                                    3   HSODRV          MODULE  MAIN, STACKSIZE(8)
                                    4
                                    5
                                    6           PUBLIC  HSO_ON_0 , HSO_OFF_0
                                    7           PUBLIC  HSO_ON_1 , HSO_OFF_1
                                    8           PUBLIC  HSO_TIME , HSO_COMMAND
                                    9           PUBLIC  SP , TIMER1 , IOSO
                                   10
                                   11   $INCLUDE(DEMO96.INC)
                              =1   12   $nolist  ;  Turn listing off for include file
                              =1   60             ;  End of include file
                                   61
    0028                           62   rseg at 28H
                                   63
                                   64           EXTRN   OLD_STAT        :byte
                                   65
    0028                           66           HSO_ON_0:       dsw     1
    002A                           67           HSO_OFF_0:      dsw     1
    002C                           68           HSO_ON_1:       dsw     1
    002E                           69           HSO_OFF_1:      dsw     1
    0030                           70           count:          dsb     1
                                   71
    2080                           72   cseg  at 2080H
                                   73
                                   74           EXTRN   wait    :entry
                                   75
    2080 FA                        76   strt:   DI
    2081 A1000118                  77           LD      SP, #100H
    2085 510F1500            E     78           ANDB    OLD_STAT, IOSO, #0FH
    2089 950F00              E     79           XORB    OLD_STAT, #0FH
                                   80
    208C                           81   initial:
    208C A1000122                  82           LD      CX, #0100H
                                   83
    2090 A100101C                  84   loop:   LD      AX, #1000H
    2094 48221C20                  85           SUB     BX, AX, CX
    2098 A0221C                    86           LD      AX, CX
                                   87
```

270061-79

```
209B C0281C                88          ST      AX, HSO_ON_0
209E C02A20                89          ST      BX, HSO_OFF_0
                           90
20A1 08011C                91          SHR     AX,#1
20A4 080120                92          SHR     BX,#1
20A7 C02C1C                93          ST      AX, HSO_ON_1
20AA C02E20                94          ST      BX, HSO_OFF_1
                           95
20AD EF0000           E    96          CALL    wait
                           97
20B0 0722                  98          INC     CX
20B2 89000F22              99          CMP     CX, #00F00H
20B6 D7D8                 100          BNE     loop
                          101
20B8 27D2                 102          BR      initial
                          103
20BA                      104          END

ASSEMBLY COMPLETED,  NO ERROR(S) FOUND.
```

270061-80

```
SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE:  :F3:HSOMOD.A96
OBJECT FILE:  :F3:HSOMOD.OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND:  NOSB

ERR LOC  OBJECT                  LINE       SOURCE STATEMENT
                                    1   $TITLE('HSOMOD.A96:  8096 PWM PROGRAM MODIFIED FOR DRIVER')
                                    2   $PAGEWIDTH(130)
                                    3
                                    4   ; This program will provide 3 PWM outputs on HSO pins 0-2
                                    5   ; The input parameters passed to the program are:
                                    6   ;
                                    7   ;                HSO_ON_N    HSO on time for pin N
                                    8   ;                HSO_OFF_N   HSO off time for pin N
                                    9
                                   10   ;        Where:   Times are in timer1 cycles
                                   11   ;                 N takes values from 0 to 3
                                   12
                                   13   ;;;
```

```
0004                            45  store_stat:
0004 510F0002           E       46          ANDB  NEW_STAT, IOSO, #0FH        ; Store new status of HSO
0008 980201             R       47          CMPB  OLD_STAT, NEW_STAT
000B DFF3                       48          JE    wait
000D 940201             R       49          XORB  OLD_STAT, NEW_STAT
                                50
                                51
0010                            52  check_0:
0010 300113             R       53          JBC   OLD_STAT, 0, check_1        ; Jump if OLD_STAT(0)=NEW_STAT(0)
0013 380209             R       54          JBS   NEW_STAT, 0, set_off_0
                                55
0016                            56  set_on_0:
0016 B13000             E       57          LDB   HSO_COMMAND, #00110000B     ; Set HSO for timer1, set pin 0
0019 44000000           E       58          ADD   HSO_TIME, TIMER1, HSO_OFF_0 ; Time to set pin = Timer1 value
001D 2007                       59          BR    check_1                     ;  + Time for pin to be low
                                60
001F                            61  set_off_0:
001F B11000             E       62          LDB   HSO_COMMAND, #00010000B     ; Set HSO for timer1, clear pin 0
0022 44000000           E       63          ADD   HSO_TIME, TIMER1, HSO_ON_0  ; Time to clear pin = Timer1 value
                                64                                            ;  + Time for pin to be high
0026                            65  check_1:
0026 310113             R       66          JBC   OLD_STAT, 1, check_done     ; Jump if OLD_STAT(1)=NEW_STAT(1)
0029 390209             R       67          JBS   NEW_STAT, 1, set_off_1
                                68
002C                            69  set_on_1:
002C B13100             E       70          LDB   HSO_COMMAND, #00110001B     ; Set HSO for timer1, set pin 1
002F 44000000           E       71          ADD   HSO_TIME, TIMER1, HSO_OFF_1 ; Time to set pin = Timer1 value
0033 2007                       72          BR    check_done
                                73
0035                            74  set_off_1:
0035 B11100             E       75          LDB   HSO_COMMAND, #00010001B     ; Set HSO for timer1, clear pin 1
0038 44000000           E       76          ADD   HSO_TIME, TIMER1, HSO_ON_1  ; Time to clear pin = Timer1 value
                                77                                            ;  + Time for pin to be high
003C                            78  check_done:
003C B00201             R       79          LDB   OLD_STAT, NEW_STAT          ; Store current status and
                                80                                            ; wait for interrupt flag
                                81
003F F0                         82          RET
                                83              use "BR  wait" if this routine is used with the driver
                                84
0040                            85          END

ASSEMBLY COMPLETED,   NO ERROR(S) FOUND.
```

270061-82

```
SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE:   F3:SP.A96
OBJECT FILE:   F3:SP.OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND:  NOSB

ERR LOC  OBJECT                  LINE        SOURCE STATEMENT
                                    1
                                    2   $TITLE('SP.A96: SERIAL PORT DEMO PROGRAM')
                                    3
                                    4   $INCLUDE(DEMO96.INC)
                               =1   5   $nolist  ;  Turn listing off for include file
                               =1  53            ;  End of include file
                                   54
    0028                           55   rseg     at 28H
                                   56
    0028                           57            CHR:     dsb      1
    0029                           58            SPTEMP:  dsb      1
    002A                           59            TEMP0:   dsb      1
    002B                           60            TEMP1:   dsb      1
    002C                           61            RCV_FLAG:        dsb     1
                                   62
    200C                           63   cseg     at 200CH
                                   64
    200C 9C20                      65            DCW      ser_port_int
                                   66
    2080                           67   cseg     at 2080H
                                   68
    2080 A1000118                  69            LD       SP, #100H
                                   70
    2084 B12016                    71            LDB      IOC1, #00100000B              ; Set P2.0 to TXD
                                   72
                                   73                     ; Baud rate = input frequency / (64*baud_val)
                                   74                     ; baud_val  = (input frequency/64) / baud rate
                                   75
                                   76
      0027                         77   baud_val         equ      39              ; 39 = (12,000,000/64)/4800 baud
                                   78
      0080                         79   BAUD_HIGH        equ      ((baud_val-1)/256) OR 80H       ; Set MSB to 1
      0026                         80   BAUD_LOW         equ      (baud_val-1) MOD 256
                                   81
                                   82
    2087 B1260E                    83            LDB      BAUD_REG, #BAUD_LOW
    208A B1800E                    84            LDB      BAUD_REG, #BAUD_HIGH
                                   85
```

270061-83

```
208D B14911                86          LDB     SPCON, #01001001B       ; Enable receiver, Mode 1
                           87
                           88                          ; The serial port is now initialized
                           89
                           90
2090 C42807                91          STB     SBUF, CHR               ; Clear serial Port
2093 B1202A                92          LDB     TEMPO, #00100000B       ; Set TI-temp
                           93
2096 B14008                94          LDB     INT_MASK, #01000000B    ; Enable Serial Port Interrupt
2099 FB                    95          EI
209A 27FE                  96  loop:   BR      loop            ; Wait for serial port interrupt
                           97
                           98
209C                       99  ser_port_int:
209C F2                   100          PUSHF
209D                      101  rd_again:                               ; This section of code can be replaced
209D B01129               102          LDB     SPTEMP, SPSTAT          ; with "ORB  TEMPO, SP_STAT" when the
20A0 90292A               103          ORB     TEMPO, SPTEMP           ; serial port TI and RI bugs are fixed
20A3 716029               104          ANDB    SPTEMP,#01100000B
20A6 D7F5                 105          JNE     rd_again        ; Repeat until TI and RI are properly cleared
                          106
20A8                      107  get_byte:
20A8 362A09               108          JBC     TEMPO, 6, put_byte      ; If RI-temp is not set
20AB C42807               109          STB     SBUF, CHR               ; Store byte
20AE 71BF2A               110          ANDB    TEMPO, #10111111B       ; CLR RI-temp
20B1 B1FF2C               111          LDB     RCV_FLAG, #0FFH         ; Set bit-received flag
                          112
20B4                      113  put_byte:
20B4 302C18               114          JBC     RCV_FLAG, 0, continue   ; If receive flag is cleared
20B7 352A15               115          JBC     TEMPO, 5, continue      ; If TI was not set
20BA B02807               116          LDB     SBUF, CHR               ; Send byte
20BD 71DF2A               117          ANDB    TEMPO, #11011111B       ; CLR TI-temp
                          118
20C0 717F28               119          ANDB    CHR, #01111111B         ; This section of code appends
20C3 990D28               120          CMPB    CHR,#0DH                ; an LF after a CR is sent
20C6 D705                 121          JNE     clr_rcv
20C8 B10A28               122          LDB     CHR, #0AH
20CB 2002                 123          BR      continue
                          124
20CD                      125  clr_rcv:
20CD 112C                 126          CLRB    RCV_FLAG                ; Clear bit-received flag
                          127
20CF                      128  continue:
20CF F3                   129          POPF
20D0 F0                   130          RET
                          131
20D1                      132          END

ASSEMBLY COMPLETED,   NO ERROR(S) FOUND.
```

270061-84

```
SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE:  F3:ATOD.A96
OBJECT FILE:  F3:ATOD.OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND:  NOSB

ERR LOC  OBJECT              LINE      SOURCE STATEMENT
                               1   $TITLE('ATOD.A96:  SCANNING THE A TO D CHANNELS')
                               2
                               3   $INCLUDE(DEMO96.INC)
                          =1   4   $nolist  ;  Turn listing off for include file
                          =1  52            ;  End of include file
                              53
     0028                     54   RSEG     at  28H
                              55
       0020                   56            BL       EQU      BX:BYTE
       001E                   57            DL       EQU      DX:BYTE
                              58
     0028                     59   RESULT_TABLE:
     0028                     60            RESULT_1:        dsw      1
     002A                     61            RESULT_2:        dsw      1
     002C                     62            RESULT_3:        dsw      1
     002E                     63            RESULT_4:        dsw      1
                              64
                              65
     2080                     66   cseg     at  2080H
                              67
                              68
     2080 A1000118            69   start:   LD       SP, #100H        ; Set Stack Pointer
     2084 0120                70            CLR      BX
                              71
     2086 55082002            72   next:    ADDB     AD_COMMAND,BL, #1000B       ; Start conversion on channel
                              73                                                ; indicated by BL register
                              74
     208A FD                  75            NOP              ; Wait for conversion to start
     208B FD                  76            NOP
     208C 3B02FD              77   check:   JBS      AD_RESULT_LO, 3, check  ; Wait while A to D is busy
                              78
     208F B0021C              79            LDB      AL, AD_RESULT_LO        ; Load low order result
     2092 B0031D              80            LDB      AH, AD_RESULT_HI        ; Load high order result
                              81
     2095 5420201E            82            ADDB     DL, BL, BL              ; DL=BL*2
     2099 AC1E1E              83            LDBZE    DX, DL
     209C C31E281C            84            ST       AX, RESULT_TABLE[DX]    ; Store result indexed by BL*2
                              85
     20A0 1720                86            INCB     BL                ; Increment BL modulo 4
```

270061-85

```
20A2 710320                87          ANDB    BL, #03H
                           88
20A5 27DF                  89          BR      next
                           90
20A7                       91          END

ASSEMBLY COMPLETED,  NO ERROR(S) FOUND.
```

270061-86

```
SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE:  :F3:A2DHSO.A96
OBJECT FILE:  :F3:A2DHSO.OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND:  NOSB

ERR LOC  OBJECT                    LINE       SOURCE STATEMENT
                                      1   $TITLE ('A2DHSO.A96:  GENERATING PWM OUTPUTS FROM A TO D INPUTS')
                                      2
                                      3   ; This program will provide 3 PWM outputs on HSO pins 0-2
                                      4   ; and one on the PWM.
                                      5   ;
                                      6   ; The PWM values are determined by the input to the A/D converter.
                                      7   ;
                                      8   ;;;
```

```
2000                                    cseg    AT 2000H

2000 8020                               DCW     start           ; Timer_ovf_int
2002 1D21                               DCW     Atod_done_int
2004 8020                               DCW     start           ; HSI_data_int
2006 CC20                               DCW     HSO_exec_int

                                        $EJECT

2080                                    cseg    AT 2080H

2080 A1000118                           start:  LD      SP, #100H       ; Set Stack Pointer
2084 011C                               CLR     AX
2086 051C                               wait:   DEC     AX              ; wait approx. 0.2 seconds for
2088 D7FC                               JNE     wait                    ; SBE to finish communications

208A 1144                               CLRB    AD_NUM

208C A1800028                           LD      PWM_TIME_1, #080H
2090 A1000148                           LD      HSO_PER, #100H
2094 A140002A                           LD      HSO_ON_0, #040H
2098 A180002C                           LD      HSO_ON_1, #080H
209C A1C0002E                           LD      HSO_ON_2, #0C0H

20A0 4500010A38                         ADD     NXT_ON_T, Timer1, #100H

20A5 B13606                             LDB     HSO_COMMAND, #00110110B     ; Set HSO for timer1, set pin 0,1
20A8 A03804                             LD      HSO_TIME, NXT_ON_T          ; with interrupt
20AB FD                                 NOP
20AC FD                                 NOP
20AD B12206                             LDB     HSO_COMMAND, #00100010B     ; Set HSO for timer1, set pin 2
20B0 643804                             ADD     HSO_TIME, NXT_ON_T          ; without interrupt

20B3 91074A                             ORB     LAST_LOAD, #00000111B   ; Last loaded value was set all pins
20B6 B10A08                             LDB     INT_MASK, #00001010B    ; Enable HSO and A/D interrupts
20B9 B10A09                             LDB     INT_PENDING, #00001010B ; Fake an A/D and HSO interrupt
20BC FB                                 EI

20BD 91010F                     loop:   ORB     Port1, #00000001B       ; set P1.0
20C0 65010040                           ADD     COUNT, #01
20C4 A40042                             ADDC    COUNT+2, zero
20C7 71FE0F                             ANDB    Port1, #11111110B       ; clear P1.0
20CA 27F1                               BR      loop

                                        $EJECT
```

270061-88

```
                        132
                        133  ;;;
```

```
211B F3                 182             POPF
211C F0                 183             RET
                        184
                        185     $EJECT
                        186
                        187     ;;;
```

APPENDIX C
SOFTWARE SERIAL PORT

```
SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE:  :F3:SWPORT.A96
OBJECT FILE:  :F3:SWPORT.OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND:  NOSB

ERR LOC  OBJECT                  LINE      SOURCE STATEMENT
                                    1   $TITLE('SWPORT.A96 : SOFTWARE IMPLEMENTED ASYNCHRONOUS SERIAL PORT')
                                    2
                                    3   ; This module provides a software implemented asynchronous serial port
                                    4   ; for the 8096.  HSO.5 is used for transmit data.  HSI.2 is used for
                                    5   ; receive data.  Note: the choice of HSO.5 and HSI.2 is arbitrary).
                                    6
                                    7   $INCLUDE(DEMO96.INC)
                               =1   8   $nolist  ;  Turn listing off for include file
                               =1  56            ;  End of include file
                                   57
                                   58   ;        VARIABLES NEEDED BY THE SOFTWARE SERIAL PORT
                                   59   ;        ===========================================
                                   60   ;
    0000                           61            rseg
                                   62
    0000                           63   ios1_save:       dsb 1     ; Used to save contents of ios1
    0001                           64   rcve_state:      dsb 1
      0001                         65     rxrdy          equ 1     ; indicates receive done
      0002                         66     rxoverrun      equ 2     ; indicates receive overflow
      0004                         67     rip            equ 4     ; receive in progress flag
    0002                           68   rcve_buf:        dsb 1     ; used to double buffer receive data
    0003                           69   rcve_reg:        dsb 1     ; used to deserialize receive
    0004                           70   sample_time:     dsw 1     ; records last receive sample time
                                   71
    0006                           72   serial_out:      dsw 1     ; Holds the output character+framing (start and
                                   73                             ;   stop bits) for transmit process.
    0008                           74   baud_count:      dsw 1     ; Holds the period of one bit in units
                                   75                             ;   of T1 ticks.
    000A                           76   txd_time:        dsw 1     ; Transition time of last Txd bit that was
                                   77                             ;   sent to the CAM
    000C                           78   char:            dsb 1     ; for test only
                                   79   ;
                                   80   ;        COMMANDS ISSUED TO THE HSO UNIT
                                   81   ;        ===============================
                                   82   ;
      0035                         83   mark_command     equ     00110101b       ; timer1,set,interrupt on 5
      0015                         84   space_command    equ     00010101b       ; timer1,clr,interrupt on 5
      0018                         85   sample_command   equ     00011000b       ; software timer 0
                                   86
                                   87   $eject
```

270061-91

```
2080                            88          cseg at 2080h
                                89  ;
2080                            90  reset_loc:
                                91  ; The 8096 starts executing here on reset, the program will initialize the
                                92  ; the software serial port and run a simple test to excercize it.
                                93  ;
2080 FA                         94          di
2081 A1F00018                  95          ld      sp,#0f0h
2085 C9C012                     96          push    #4800
2088 EF0000             R       97          call    setup_serial_port
208B B16C08                     98          ldb     int_mask,#01101100b     ; serial, swt,hso,hsi
208E FB                         99          ei
                               100
                               101  ;
208F                           102  test1:
                               103  ; A simple test of the serial port routines.
                               104  ; While no characters are received an incrementing pattern is sent to the
                               105  ; serial output.  When a character is received the incrementing pattern
                               106  ; "jumps" to the character receved and proceeds from there.
                               107  ;
  000D                         108          CR      equ     0DH                     ; Carriage return
208F B10D0C             R      109          ldb     char,#CR
2092                           110  testiloop:
2092 AC0C1C             R      111          ldbze   ax,char
2095 C81C                      112          push    ax
2097 EF3000             R      113          call    char_out
                               114
209A 990D0C             R      115          cmpb    char,#CR                ; Pause on Carriage return
209D D706                      116          bne     nopause
209F 011C                      117          clr     ax
20A1                           118  pause:
20A1 071C                      119          inc     ax
20A3 D7FC                      120          bne     pause
20A5                           121  nopause:
                               122
20A5 170C               R      123          incb    char
20A7                           124  test2:
20A7 EF4400             R      125          call    csts                    ; char ready?
20AA 98001C                    126          cmpb    al,0
20AD DFE3                      127          be      testiloop               ;   loop if not
20AF EF4C00             R      128          call    char_in
20B2 B01C0C             R      129          ldb     char,al
20B5 27DB                      130          br      testiloop
                               131  $eject
```

270061-92

```
                                132
0000                            133             cseg
                                134     ,
0000                            135     setup_serial_port:
                                136     ; Called on system reset to intiate the software serial port.
                                137     ;
0000 CC22                       138             pop     cx                      ; the return address
0002 CC20                       139             pop     bx                      ; the baud rate (in decimal)
0004 A107001E                   140             ld      dx,#0007h               ; dx:ax =500,000 (assumes 12 Mhz crystal)
0008 A120A11C                   141             ld      ax,#0A120h
000C BC201C                     142             divu    ax,bx                   ; calculate the baud count (500,000/baudrate)
000F C0081C             R       143             st      ax,baud_count
0012 C00600             R       144             st      0,serial_out            ; clear serial out
0015 B16016                     145             ldb     ioc1,#01100000b         ; Enable HSO.5 and Txd
0018 3E15FD                     146             bbs     ios0,6,$                ; Wait for room in the HSO CAM
                                147                                             ; and issue a MARK command.
001B 44140A0A           R       148             add     txd_time,timer1,20
001F B13506                     149             ldb     hso_command,#mark_command
0022 A00A04             R       150             ld      hso_time,txd_time
0025 1102               R       151             clrb    rcve_buf                ; clear out the receive variables
0027 1103               R       152             clrb    rcve_reg
0029 1101               R       153             clrb    rcve_state
002B EF4800                     154             call    init_receive            ; setup to detect a start bit
002E E322                       155             br      [cx]                    ; return
                                156     $eject
                                157     ,
0030                            158     char_out:
                                159     ; Output character to the software serial port
                                160     ;
0030 CC22                       161             pop     cx                      ; the return address
0032 CC20                       162             pop     bx                      ; the character for output
0034 B10121                     163             ldb     (bx+1),#01h             ; add the start and stop bits
0037 642020                     164             add     bx,bx                   ;   to the char and leave as 16 bit
003A                            165     wait_for_xmit:
003A 880006             R       166             cmp     serial_out,0            ; wait for serial_out=0 (it will be cleared by
003D D7FB                       167             bne     wait_for_xmit           ;   the hso interrupt process)
003F C00620             R       168             st      bx,serial_out           ; put the formatted character in serial_out
0042 E322                       169             br      [cx]                    ; return to caller
                                170     ;
0044                            171     csts:
                                172     ; Returns "true" (ax<>0) if char_in has a character.
                                173     ;
0044 011C                       174             clr     ax
0046 300102             R       175             bbc     rcve_state,0,csts_exit
0049 071C                       176             inc     ax
004B                            177     csts_exit:
004B F0                         178             ret
                                179     ;
004C                            180     char_in:
```

270061-93

```
                        181   ; Get a character from the software serial port
                        182   ;
                        183                                   ; wait for character ready
004C 3001FD          R  184           bbc     rcve_state,0,char_in
004F F2                 185           pushf                   ; set up a critical region
0050 71FE01          R  186           andb    rcve_state,#not(rxrdy)
0053 AC021C          R  187           ldbze   al,rcve_buf
0056 F3                 188           popf                    ; leave the critical region
0057 F0                 189           ret
                        190   $eject
                        191   ;
0058                    192   hso_isr:
                        193   ; Fields the hso interrupts and performs the serialization of the data.
                        194   ; Note:  this routine would be incorporated into the hso service strategy
                        195   ;        for an actual system.
                        196
2006                    197           cseg    at 2006h
2006 5800            R  198           dcw     hso_isr         ; Set up vector
                        199
0058                    200           cseg
0058 F2                 201           pushf
0059 64080A          R  202           add     txd_time,baud_count
005C 880006          R  203           cmp     serial_out,0    ; if character is done send a mark
005F DF0D               204           be      send_mark
0061 080106          R  205           shr     serial_out,#1   ; else send bit 0 of serial_out and shift
0064 DB08               206           bc      send_mark       ;   serial_out left one place.
0066                    207   send_space:
0066 B11506             208           ldb     hso_command,#space_command
0069 A00A04          R  209           ld      hso_time,txd_time
006C 2006               210           br      hso_isr_exit
006E                    211   send_mark:
006E B13506             212           ldb     hso_command,#mark_command
0071 A00A04          R  213           ld      hso_time,txd_time
                        214
0074                    215   hso_isr_exit:
0074 F3                 216           popf
0075 F0                 217           ret
                        218   $eject
                        219   ;
0076                    220   init_receive:
                        221   ; Called to prepare the serial input process to find the leading edge of
                        222   ; a start bit.
                        223   ;
0076 B10015             224           ldb     ioc0,#00000000b         ; disconnect change detector
0079 B12003             225           ldb     hsi_mode,#00100000b     ; negative edges on HSI.2
007C                    226   flush_fifo:
007C 901600          R  227           orb     ios1_save,ios1
007F 37000B          R  228           bbc     ios1_save,7,flush_fifo_done
0082 B0061C             229           ldb     al,hsi_status
0085 A0041C             230           ld      ax,hsi_time             ; trash the fifo entry
```

270061-94

```
0088 717F00        R     231             andb    ios1_save,#not(80h)     ; clear bit 7.
008B 27EF                232             br      flush_fifo
008D                     233     flush_fifo_done:
008D B11015              234             ldb     ioc0,#00010000b         ; connect HSI.2 to detector
0090 F0                  235             ret
                         236
                         237
                         238     ;
0091                     239     hsi_isr:
                         240     ; Fields interrupts from the HSI unit, used to detect the leading edge
                         241     ; of the START bit
                         242     ; Note: this routine would be incorporated into the HSI strategy of an actual
                         243     ; system.
                         244     ;
2004                     245             cseg at 2004h
2004 9100          R     246             dcw     hsi_isr                 ; setup the interrupt vector
                         247
0091                     248             cseg
0091 F2                  249             pushf
0092 C81C                250             push    ax
0094 B0061C              251             ldb     al,hsi_status
0097 A00404        R     252             ld      sample_time,hsi_time
009A 341C15              253             bbc     al,4,exit_hsi
009D 3F15FD              254             bbs     ios0,7,$                ; wait for room in HSO holding reg
00A0 A0081C        R     255             ld      ax,baud_count           ; send out sample command in 1/2
00A3 08011C              256             shr     ax,#1                   ; bit time
00A6 641C04        R     257             add     sample_time,ax
00A9 B11B06              258             ldb     hso_command,#sample_command
00AC C00404        R     259             st      sample_time,hso_time
00AF B10015              260             ldb     ioc0,#00000000b         ; disconnect hsi.2 from change detector
00B2                     261     exit_hsi:
00B2 CC1C                262             pop     ax
00B4 F3                  263             popf
00B5 F0                  264             ret
                         265     $eject
                         266     ;
00B6                     267     software_timer_isr:
                         268     ; Fields the software timer interrupt, used to deserialize the incomming data.
                         269     ; Note: this routine would be incorporated into the software timer stategy
                         270     ; in an actual system.
                         271     ;
200A                     272             cseg at 200ah
200A B600          R     273             dcw     software_timer_isr      ; setup vector
                         274
00B6                     275             cseg
00B6 F2                  276             pushf
00B7 901600        R     277             orb     ios1_save,ios1
00BA 71FE00        R     278             andb    ios1_save,#not(01h)     ; clear bit 0
00BD 51FC0100      R     279             andb    0,rcve_state,#0fch      ; All bits except rxrdy and overrun=0
00C1 D70C                280             bne     process_data
```

270061-95

```
00C3                            281   process_start_bit:
00C3 350604                     282           bbc     hsi_status,5,start_ok
00C6 2FAE                       283           call    init_receive
00C8 2032                       284           br      software_timer_exit
00CA                            285   start_ok:
00CA 910401             R       286           orb     rcve_state,#rip ; set receive in progress flag
00CD 2021                       287           br      schedule_sample
                                288
00CF                            289   process_data:
00CF 3F010E             R       290           bbs     rcve_state,7,check_stopbit
00D2 180103             R       291           shrb    rcve_reg,#1
00D5 350603                     292           bbc     hsi_status,5,datazero
00D8 918003             R       293           orb     rcve_reg,#80h   ; set the new data bit
00DB                            294   datazero:
00DB 751001             R       295           addb    rcve_state,#10h ; increment bit count
00DE 2010                       296           br      schedule_sample
                                297
00E0                            298   check_stopbit:
00E0 3506FD                     299           bbc     hsi_status,5,$  ; DEBUG ONLY
00E3 B00302             R       300           ldb     rcve_buf,rcve_reg
00E6 910101             R       301           orb     rcve_state,#rxrdy
00E9 710301             R       302           andb    rcve_state,#03h ; Clear all but ready and overrun bits
00EC 2F88                       303           call    init_receive
00EE 200C                       304           br      software_timer_exit
                                305
00F0                            306   schedule_sample:
00F0 3F15FD                     307           bbs     ios0,7,$        ; wait for holding reg empty
00F3 B11806                     308           ldb     hso_command,#sample_command
00F6 640804             R       309           add     sample_time,baud_count
00F9 C00404             R       310           st      sample_time,hso_time
                                311
00FC                            312   software_timer_exit:
00FC F3                         313           popf
00FD F0                         314           ret
                                315
                                316
00FE                            317           end

ASSEMBLY COMPLETED,   NO ERROR(S) FOUND
```

270061-96

```
SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE:  :F3:MOTCON.A96
OBJECT FILE:  :F3:MOTCON.OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND:  NOSB

ERR LOC  OBJECT           LINE       SOURCE STATEMENT
                             1   $TITLE ('MOTCON.A96:  Motor Control Example Program')
                             2
                             3   ;       USE WITH C-STEP or later parts
                             4
                             5   ;                                          December 20, 1984
                             6
                             7   $INCLUDE(DEMO96.INC)
                       =1    8   $nolist  ;  Turn listing off for include file
                       =1   56           ;  End of include file
                            57
                            58   ;;;;;;;;;;      Initial Values
                            59
     001E                   60   min_hsi1_t      equ     30  ; min period for PHA edges in mode1 before mode2
                            61
     003C                   62   min_hsi_t       equ     2*min_hsi1_t
                            63                           ; min period for PHA edges in mode0 before mode1
                            64
     0069                   65   max_hsi1_t      equ     3*min_hsi1_t + min_hsi1_t/2
                            66                           ; max period for PHA edges in mode1 before mode0
                            67
                            68
     006E                   69   HSO0_dly_period equ     110     ; delay for HSO timer 0 (timed count of pulses)
                            70                           ; min period for 5 T2 clocks before mode 1
                            71
     00FA                   72   swt1_dly_period equ     250     ; delay for software timer 1
     00FA                   73   swt2_dly_period equ     250     ; delay for software timer 2
     00FF                   74   max_power       equ     0ffh
     00FF                   75   max_brake       equ     0ffh
     0080                   76   maximum_hold    equ     080H
     04B0                   77   brake_pnt       equ     1200
     0064                   78   position_pnt    equ     100
     0010                   79   velocity_pnt    equ     16
                            80
                            81
     0024                   82   RSEG at 024H
                            83
     0024                   84        tmp:       dsl 1
     0028                   85        timer_2:   dsl 1
```

270061-97

```
002C                    86          tmr2_old:       dsl 1
0030                    87          position:       dsl 1
0034                    88          des_pos:        dsl 1
0038                    89          pos_err:        dsl 1
003C                    90          delta_p:        dsl 1
0040                    91          time:           dsl 1
0044                    92          des_time:       dsl 1
0048                    93          time_err:       dsl 1
                        94
                        95  $EJECT
                        96
004C                    97          last_time_err   dsw 1
004E                    98          last_pos_err:   dsw 1
0050                    99          pos_delta:      dsw 1
0052                   100          time_delta:     dsw 1
0054                   101          last_pos:       dsw 1
0056                   102          last1_time:     dsw 1
0058                   103          last2_time:     dsw 1
005A                   104          boost:          dsw 1
005C                   105          tmp1:           dsw 1
005E                   106          out_ptr:        dsw 1
0060                   107          offset:         dsw 1
0062                   108          nxt_pos:        dsw 1
0064                   109          rpwr:           dsw 1
0066                   110          old_t2:         dsw 1
                       111
0068                   112          direct:         dsb 1   ;  1=forward, 0=reverse
0069                   113          pwm_dir:        dsb 1
006A                   114          hsi_s0:         dsb 1
006B                   115          last_stat:      dsb 1
006C                   116          pwm_pwr:        dsb 1
006D                   117          ios1_bak:       dsb 1
006E                   118          TR_COL:         DSB 1   ; COLLECT TRACE IF TR_COL=00
006F                   119          main_dly:       dsb 1
                       120
0070                   121          max_pwr:        dsw 1
0072                   122          max_brk:        dsw 1
0074                   123          max_hold:       dsw 1
0076                   124          vel_pnt:        dsw 1
0078                   125          brk_pnt:        dsw 1
007A                   126          pos_pnt:        dsw 1
007C                   127          HSO0_dly:       dsw 1
007E                   128          swt1_dly:       dsw 1
0080                   129          swt2_dly:       dsw 1
0082                   130          min_hsi:        dsw 1
0084                   131          min_hsi1:       dsw 1
0086                   132          max_hsi1:       dsw 1
                       133
                       134
0100                   135  dseg at 100H
```

270061-98

```
                        136
0100                    137 mode_view:      dsb     1
0102                    138 count_out:      dsw     1
0104                    139 err_view:       dsw     1
                        140
                        141
                        142 $eject
                        143
                        144 ;       PIN#    PORT    FLAG USAGE
                        145
                        146 ;       22      P1.0    mode0 0   mode1 1   mode2 1  or 0
                        147 ;       23      P1.1          0         0         1     1
                        148 ;       24      P1.2    software timer 2 routine enter/leave
                        149 ;       25      P1.3    Main program toggle
                        150 ;       26      P1.4    HSI overflow toggle
                        151 ;       37      P1.5    software timer 0 routine enter/leave
                        152 ;       38      P1.6    hsi_int enter/leave
                        153 ;       39      P1.7    software timer 1 routine enter/leave
                        154 ;       40      P2.6    Input direction (0=reverse, 1=forward)
                        155 ;       45      P2.7    direction 0=rev, 1=fwd
                        156
2000                    157 cseg    at      2000H
2000 0022               158         dcw     timer_ovf_int
2002 1020               159         dcw     atod_done_int
2004 0424               160         dcw     hsi_data_int
2006 8022               161         dcw     hso_exec_int
2008 1020               162         dcw     hsi_0_int
200A 2022               163         dcw     soft_tmr_int
200C 1020               164         dcw     ser_port_int
200E 1020               165         dcw     external_int
                        166
2010                    167 atod_done_int:
2010                    168 hsi_0_int:
2010                    169 ser_port_int:
2010                    170 external_int:
                        171
2080                    172 cseg    at      2080H
                        173
2080 A1F00018           174 init:   ld      sp,#0F0H
2084 B1FF17             175         ldb     pwm_control,#0FFH
                        176
2087 1168               177         clrb    direct
2089 A170175C           178         ld      tmp1,#6000          ; wait about 3 seconds for motor
208D 055C               179 delay:  dec     tmp1            ;   to come to a stop
208F E068FD             180         djnz    direct,$            ; wait 0.512 milliseconds
2092 88005C             181         cmp     tmp1,zero
2095 D2F6               182         jgt     delay
                        183
2097 B1FF0F             184         ldb     port1,#0FFH
209A B1FF10             185         ldb     port2,#0ffH
```

270061-99

```
209D B12516          186             ldb     IOC1,#00100101B ; Disable HSO.4,HSO.5, HSI_INT=first,
                     187                                     ; Enable PWM,TXD,TIMER1_OVRFLOW_INT
                     188
20A0 71FC0F          189             andb    Port1,#11111100B        ; clear P1.0,1 (set mode 0)
20A3 B19903          190             ldb     HSI_mode,#10011001B     ; set hsi.1,3 -; hsi.0,2 +
20A6 B15715          191             ldb     IOC0,#01010111B         ; Enable all hsi
                     192                                             ; T2 CLOCK=T2CLK, T2RST=T2RST
                     193                                             ; Clear timer2
                     194     $eject
                     195
20A9 A00400          196             ld      zero,hsi_time
20AC 0140            197             clr     time
20AE 0142            198             clr     time+2
20B0 0128            199             clr     timer_2
20B2 012A            200             clr     timer_2+2
20B4 0130            201             clr     position
20B6 0132            202             clr     position+2
20B8 0154            203             clr     last_pos
20BA 0134            204             clr     des_pos
20BC 0136            205             clr     des_pos+2
20BE 0144            206             clr     des_time
20C0 0146            207             clr     des_time+2
20C2 A00A56          208             ld      last1_time,Timer1
20C5 4900085658      209             sub     last2_time,last1_time,#800H
20CA 116D            210             clrb    ios1_bak
20CC 1109            211             clrb    int_pending
20CE A1F0015E        212             ld      out_ptr,#1F0H
20D2 A13C0082        213             ld      min_hsi,#min_hsi_t
20D6 A11E0084        214             ld      min_hsi1,#min_hsi1_t
20DA A1690086        215             ld      max_hsi1,#max_hsi1_t
20DE A16E007C        216             ld      HSO0_dly,#HSO0_dly_period
20E2 A1FA007E        217             ld      swt1_dly,#swt1_dly_period
20E6 A1FA0080        218             ld      swt2_dly,#(swt2_dly_period)
20EA A1FF0070        219             ld      max_pwr,#max_power
20EE A1FF0072        220             ld      max_brk,#max_brake
20F2 A1B00074        221             ld      max_hold,#maximum_hold
20F6 A1B00478        222             ld      brk_pnt,#brake_pnt
20FA A164007A        223             ld      pos_pnt,#position_pnt
20FE A1100076        224             ld      vel_pnt,#velocity_pnt
2102 A1002962        225             ld      nxt_pos,#pos_table
2106 B0006C          226             ldb     pwm_pwr,zero
2109 B10169          227             ldb     pwm_dir,#01h            ; FORWARD
                     228
210C B12D08          229             ldb     int_mask,#00101101B     ; Enable tmr_ovf, hsi, swt, HSO, interrupts
210F B13006          230             ldb     hso_command,#30H        ; set HSO_0
2112 447C0A04        231             add     hso_time,timer1,HSO0_dly
2116 FD              232             nop
2117 FD              233             NOP
2118 B13906          234             ldb     hso_command,#39H        ; set swt_1
211B 447E0A04        235             add     hso_time,timer1,swt1_dly
```

270061-A0

```
211F FD                 236             nop
2120 FD                 237             nop
2121 B13A06             238             ldb     hso_command,#3AH        ; set swt_2
2124 44800A04           239             add     hso_time,timer1,swt2_dly
                        240
2128 A00A40             241             ld      time,TIMER1
212B A00C2C             242             ld      tmr2_old,timer2
212E FB                 243             ei
                        244
212F E7CE06             245             br      main_prog
                        246
                        247     $eject
                        248
                        249
                        250     ;;;
```

```
2230 EFCD03                     call    swt1_expired
2233                    chk_swt2:
2233 326D06                     jbc     ios1_bak,2,chk_swt3
2236 71FB6D                     andb    ios1_bak,#11111011B     ; Clear bit 2
2239 EF4401                     call    swt2_expired
223C                    chk_swt3:
223C 346D03                     jbc     ios1_bak,4,swt_int_done
223F 71F76D                     andb    ios1_bak,#11110111B     ; Clear bit 3
                        ;       call    swt3_expired
2242                    swt_int_done:
2242 F3                         popf
2243 F0                         ret     ; END OF SOFTWARE TIMER INTERRUPT ROUTINE

                        $eject
                        ;;;
```

```
22B3 643C30          336  in_fwd:  add     position,delta_p
22B6 A40032          337           addc    position+2,zero
22B9 2006            338           br      chk_mode
                     339
22BB 683C30          340  in_rev:  sub     position,delta_p
22BE A80032          341           subc    position+2,zero
                     342
22C1                 343  chk_mode:
22C1 4866285C        344           sub     tmp1,Timer_2,old_t2     ; Check count difference in tmp1
22C5 8905005C        345           cmp     tmp1,#5                 ; set mode1 if count is too low
22C9 D21C            346           jgt     end_swt0                ; count <= 5
                     347
22CB                 348  set_mode1:
22CB 71FD0F          349           andb    Port1,#11111101B        ; Clear P1.1, set P1.0 (set mode 1)
22CE 91010F          350           orb     Port1,#00000001B
22D1 B10515          351           ldb     IOC0,#00000101B         ; enable HSI 0 and 1
22D4 A00400          352           ld      zero, HSI_TIME
22D7 48840A56        353           sub     last1_time,Timer1,min_hsi1
                     354                   ; set up so (time-last2_time)>min_hsi1 on next HSI
                     355  $EJECT
                     356
22DB                 357  clr_hsi:
22DB A00400          358           ld      ZERO, HSI_TIME
22DE 717F6D          359           andb    ios1_bak,#01111111B             ; clear bit 7
22E1 90166D          360           orb     ios1_bak,ios1
22E4 3F6DF4          361           jbs     ios1_bak,7,clr_hsi      ; If hsi is triggered then clear hsi
                     362
22E7                 363  end_swt0:
22E7 A02866          364           ld      old_t2,TIMER_2
22EA 71DF0F          365           andb    port1,#11011111B        ; clear P1.5
22ED F3              366           POPF
22EE F0              367           ret
                     368
                     369
                     370
                     371  ;;;
```

```
                        386
2395                    387  pulsing:
2395 306E0C             388          jbc     tr_col,0,swt2_done
                        389
2398 C25F32             390          st      position+2,[out_ptr]+   ;  position high, position low
239B C25F30             391          st      position,[out_ptr]+
                        392
239E C25F68             393          st      direct,[out_ptr]+
23A1 C25F6C             394          st      pwm_pwr,[out_ptr]+
                        395
                        396                                          ; store 8 bytes externally
                        397
23A4                    398  swt2_done:
23A4 48560A5C           399          sub     tmp1,timer1,last1_time
23A8 8900185C           400          cmp     tmp1,#1800H
23AC D104               401          jnh     swt2_ret        ; keep (Timer1-last1_time)<2000H
                        402
23AE 65001056           403          add     last1_time,#1000H
23B2                    404  swt2_ret:
23B2 71FB0F             405          andb    port1,#11111011B        ; clear port1.2
23B5 F3                 406          popf
23B6 F0                 407          ret
                        408
                        409  $EJECT
                        410  ;;;
```

```
2421 3A6A2C          436            jbs     hsi_s0,2,a_fall
2424 3C6A4D          437            jbs     hsi_s0,4,b_rise
2427 3E6A5A          438            jbs     hsi_s0,6,b_fall
242A 2094            439            br      no_cnt
                     440
242C A05658          441    a_rise: ld      last2_time,last1_time
242F A04056          442            ld      last1_time,time
2432 685840          443            sub     time,last2_time
2435 888240          444            cmp     time,min_hsi
2438 D906            445            jh      tst_statr
                     446    ;set mode1-
243A 91010F          447            orb     Port1,#00000001B        ; Set P1.0 (in mode 1)
243D B10515          448            ldb     IOC0,#00000101B         ; Enable HSI 0 and 1
2440                 449    tst_statr:
2440 3E6B5B          450            jbs     last_stat,6,going_fwd
2443 3C6B67          451            jbs     last_stat,4,going_rev
2446 3A6B50          452            jbs     last_stat,2,change_dir
2449 98006B          453            cmpb    last_stat,zero
244C DF46            454            je      first_time              ; first time in mode0
244E 27B2            455            br      no_int1
                     456
2450 A05658          457    a_fall: ld      last2_time,last1_time
2453 A04056          458            ld      last1_time,time
2456 685840          459            sub     time,last2_time
2459 888240          460            cmp     time,min_hsi
245C D906            461            jh      tst_statf
                     462    ;set mode1-
245E 91010F          463            orb     Port1,#00000001B        ; Set P1.0 (in mode 1)
2461 B10515          464            ldb     IOC0,#00000101B         ; Enable HSI 0 and 1
                     465    $EJECT
2464                 466    tst_statf:
2464 3C6B37          467            jbs     last_stat,4,going_fwd
2467 3E6B43          468            jbs     last_stat,6,going_rev
246A 386B2C          469            jbs     last_stat,0,change_dir
246D 98006B          470            cmpb    last_stat,zero
2470 DF22            471            je      first_time              ; first time in mode0
2472 2057            472            br      no_int
                     473
2474 386B27          474    b_rise: jbs     last_stat,0,going_fwd
2477 3A6B33          475            jbs     last_stat,2,going_rev
247A 3E6B1C          476            jbs     last_stat,6,change_dir
247D 98006B          477            cmpb    last_stat,zero
2480 DF12            478            je      first_time              ; first time in mode0
2482 2047            479            br      no_int
                     480
2484 3A6B17          481    b_fall: jbs     last_stat,2,going_fwd
2487 386B23          482            jbs     last_stat,0,going_rev
248A 3C6B0C          483            jbs     last_stat,4,change_dir
248D 98006B          484            cmpb    last_stat,zero
2490 DF02            485            je      first_time              ; first time in mode0
```

270061-A5

```
2492 2037                486             br      no_int
                         487
2494                     488     first_time:
2494 C46B6A              489             stb     hsi_s0,last_stat
2497 2072                490             br      done_chk        ; add delta position
                         491
                         492
2499                     493     change_dir:
2499 1268                494             notb    direct
249B 30680F              495     no_inc: jbc     direct,0,going_rev
                         496
249E                     497     going_fwd:
249E 914010              498             orb     PORT2,#01000000B        ; set P2.6
24A1 B10168              499             ldb     direct,#01              ; direction = forward
24A4 65010030            500             add     position,#01
24A8 A40032              501             addc    position+2,zero
24AB 200D                502             br      st_stat
24AD                     503     going_rev:
24AD 71BF10              504             andb    PORT2,#10111111B        ; clear P2.6
24B0 B10068              505             ldb     direct,#00              ; direction = reverse
24B3 69010030            506             sub     position,#01
24B7 A80032              507             subc    position+2,zero
                         508
24BA                     509     st_stat:
24BA C46B6A              510             stb     hsi_s0,last_stat
24BD                     511     load_lasts:
24BD A0282C              512             ld      tmr2_old,timer_2
24C0 717F6D              513     no_cnt: andb    ios1_bak,#01111111B     ; clr bit 7
24C3 90166D              514             orb     ios1_bak,ios1
24C6 376D02              515             jbc     ios1_bak,7,no_int
24C9 2746                516     again:  br      get_values
                         517
24CB 71BF0F              518     no_int: andb    port1,#10111111B        ; Clear P1.6
24CE F3                  519             popf
24CF F0                  520             ret     ; end of hsi_data interrupt routine
                         521             ; Routine for mode 1 follows and then returns to "load_lasts"
                         522     $EJECT
                         523
                         524
24D0                     525     In_mode_1               ; mode 1 HSI routine
                         526
24D0 51506A5C            527             andb    tmp1,hsi_s0,#01010000B
24D4 D7EA                528             jne     no_cnt
24D6                     529     cmp_time:                               ; Procedure which sets mode 1 also
                         530                                             ; sets times to pass the tests
24D6 A05658              531             ld      last2_time,last1_time
24D9 A04056              532             ld      last1_time,time
                         533
24DC 4858405C            534     cmp1:   sub     tmp1,time,last2_time
24E0 88845C              535             cmp     tmp1,min_hsi1
```

270061-A6

```
24E3 D914                536             jh      check_max_time
                         537
24E5                     538     set_mode_2:
24E5 91020F              539             orb     Port1,#00000010B        ; Set P1.1 (in mode 2)
24E8 B10015              540             ldb     IOC0,#00000000B         ; Disable all HSI
24EB A00400              541     mt_hsi: ld      zero,hsi_time           ; empty the hsi fifo
24EE 717F6D              542             andb    ios1_bak,#01111111B             ; clear bit 7
24F1 90166D              543             orb     ios1_bak,ios1
24F4 3F6DF4              544             jbs     ios1_bak,7,mt_hsi       ; If hsi is triggered then clear hsi
24F7 2012                545             br      done_chk
                         546
24F9                     547     check_max_time:
24F9 4858405C            548             sub     tmp1,time,last2_time
24FD 88865C              549             cmp     tmp1,max_hsi1           ; max_hsi = addition to min_hsi for
                         550                                             ; total time
2500 D109                551             jnh     done_chk
                         552
2502                     553     set_mode_0:
2502 71FC0F              554             andb    Port1,#11111100B        ; clear P1.0,1  set mode 00
2505 B15515              555             ldb     IOC0,#01010101B         ; Enable all HSI
2508 B0006B              556             ldb     last_stat,zero
                         557
250B                     558     done_chk:
250B 482C283C            559             sub     delta_p,timer_2,tmr2_old        ; get timer2 countidifference
250F 306808              560             jbc     direct,0,add_rev
2512                     561     add_fwd:
2512 643C30              562             add     position,delta_p
2515 A40032              563             addc    position+2,zero
2518 27A3                564             br      load_lasts
251A                     565     add_rev:
251A 683C30              566             sub     position,delta_p
251D A80032              567             subc    position+2,zero
2520 279B                568             br      load_lasts
                         569
                         570     $eject
                         571     ;;;
```

```
                        586
260E A0464A             587             ld      time_err+2,des_time+2   ; Calculate time & position error
2611 A0363A             588             ld      pos_err+2,des_pos+2
2614 4840444B           589             sub     time_err,des_time,time          ; values are set
2618 A8424A             590             subc    time_err+2,time+2
261B 48303438           591             sub     pos_err,des_pos,position
261F A8323A             592             subc    pos_err+2,position+2
                        593
2622 FB                 594             EI
                        595
2623 48484C52           596             sub     time_delta,last_time_err,time_err
2627 A0484C             597             ld      last_time_err,time_err
                        598
262A 48384E50           599             sub     pos_delta,last_pos_err,pos_err
262E A0384E             600             ld      last_pos_err,pos_err
                        601
                        602             ;;;;;   Time_err = Desired time to finish - current time
                        603             ;;;;;   Pos_err  = Desired position to finish - current position
                        604             ;;;;;   Pos_delta  = Last position error - Curent position error
                        605             ;;;;;   Time_delta = Last time error - Current time error
                        606             ;;;;;      note that errors should get smaller so deltas will be
                        607             ;;;;;      positive for forward motion (time is always forward)
                        608
                        609
2631                    610     chk_dir:
2631 88003A             611             cmp     pos_err+2,zero
2634 D60D               612             jge     go_forward
                        613
2636                    614     go_backward:
2636 0338               615             neg     pos_err                 ; Pos_err = ABS VAL (pos_err)
2638 B10069             616             ldb     pwm_dir,#00h
263B 89FFFF3A           617             cmp     pos_err+2,#0ffffH
263F D70A               618             jne     ld_max
2641 200D               619             br      chk_brk
                        620
2643                    621     go_forward:
2643 B10169             622             ldb     pwm_dir,#01H
2646 88003A             623             cmp     pos_err+2,zero
2649 DF05               624             je      chk_brk
                        625     $EJECT
                        626
264B B0706C             627     ld_max: ldb     pwm_pwr,max_pwr
264E 2051               628             br      chk_sanity
                        629
2650                    630     Chk_brk:                        ;  Position_Error now = ABS(pos_err)
2650 887A38             631             cmp     pos_err,pos_pnt
2653 D11E               632             jnh     hold_position   ; position_error<position_control_point
2655 887838             633             cmp     pos_err,brk_pnt
```

270061-A8

```
2658 D9F1                634             jh      ld_max          ; position_error>brake_point
                         635
265A                     636     braking:
265A 880050              637             cmp     pos_delta,zero
265D D602                638             jge     chk_delta
265F 0350                639             neg     pos_delta
2661                     640     chk_delta:
2661 887650              641             cmp     pos_delta,vel_pnt       ; velocity = pos_delta/sample_time
2664 D10D                642             jnh     hold_position           ; jmp if  ABS(velocity) < vel_pnt
                         643
2666 B0726C              644     brake:  ldb     pwm_pwr,max_brk
2669 B06824              645             ldb     tmp,direct              ; If braking apply power in opposite
266C 1224                646             notb    tmp                     ; direction of current motion
266E B02469              647             ldb     pwm_dir,tmp
                         648
2671 2030                649             br      ld_pwr
                         650
2673                     651     Hold_position:                  ; position hold mode
2673 89020038            652             cmp     pos_err,#02
2677 D906                653             jh      calc_out        ; if position error < 2 then turn off power
2679 0126                654             clr     tmp+2
267B 015A                655             clr     boost
267D 201F                656             BR      output
                         657
267F                     658     calc_out:
267F 5DFF7424            659             mulub   tmp,max_hold,#255
2683 6C3824              660             mulu    tmp,pos_err             ; Tmp = pos_err * max_hold
2686 880050              661             cmp     pos_delta,zero
2689 D709                662             jne     no_bst
268B 6504005A            663             add     boost,#04               ; Boost is integral control
268F 645A26              664             add     tmp+2,boost             ; TMP+2 = MSB(pos_err*max_hold)
2692 2002                665             br      ck_max
2694 015A                666     no_bst: clr     boost
2696 887426              667     ck_max: cmp     tmp+2,max_hold
2699 D103                668             jnh     output
269B A07426              669     maxed:  ld      tmp+2,max_hold
269E B0266C              670     output: ldb     pwm_pwr,tmp+2
                         671
                         672
26A1                     673     chk_sanity:
26A1 2000                674             br      ld_pwr
                         675     ;;
                         676     ;;
                         677     $EJECT
                         678
26A3                     679     ld_pwr:
26A3 B06C64              680             ldb     rpwr,pwm_pwr
26A6 1264                681             notb    rpwr
26A8 38690A              682             jbs     pwm_dir,0,p2fwd
                         683
```

270061-A9

```
26AB FA          684  p2bkwd: DI
26AC 717F10      685          andb    port2,#01111111B        ; clear P2.7
26AF B06417      686          ldb     pwm_control,rpwr
26B2 FB          687          EI
26B3 2008        688          br      pwrset
26B5 FA          689  p2fwd:  DI
26B6 918010      690          orb     port2,#10000000B        ; set P2.7
26B9 B06417      691          ldb     pwm_control,rpwr
26BC FB          692          EI
                 693
26BD             694  pwrset:
26BD 88004A      695          cmp     time_err+2,zero  ; do pos_table when err is negative
26C0 D225        696          jgt     end_p
                 697  ;;;     br      end_p
                 698
26C2 89202962    699          cmp     nxt_pos,#(32+pos_table)
26C6 DE06        700          jlt     get_vals                ; jump if lower
26C8 A1002962    701          ld      nxt_pos,#pos_table
26CC 0142        702          clr     time+2
26CE             703  get_vals:
                 704
26CE A26334      705          ld      des_pos,[nxt_pos]+
26D1 A26336      706          ld      des_pos+2,[nxt_pos]+
26D4 A26346      707          ld      des_time+2,[nxt_pos]+
26D7 A26370      708          ld      max_pwr,[nxt_pos]+
26DA A07072      709          ld      max_brk,max_pwr
26DD 646034      710          add     des_pos,offset
26E0 A40036      711          addc    des_pos+2,zero
26E3 4830344E    712          sub     last_pos_err,des_pos,position
                 713
26E7 717F0F      714  end_p:  andb    port1,#01111111B        ; clear P1.7
                 715
26EA F3          716          popf
26EB F0          717          ret
                 718
                 719  $EJECT
                 720
                 721  ;;;
```

```
                        734   control:
280F                    735           orb     int_mask,#00101101B     ; enable hsi, hso, swt, tovf interrupts
280F 912D08             736           nop
2812 FD                 737           nop
2813 FD                 738           nop
2814 FD                 739           djnz    main_dly,$
2815 E06FFD             740           nop
2818 FD                 741           xorb    port1,#00001000B        ; compliment p1.3
2819 95080F             742           BR      MAIN_PROG
281C 27E2               743
                        744
                        745           CSEG AT 2900H
2900                    746
                        747   pos_table:
2900                    748
2900 00000000           749           dcl     00000000H       ; position 0
2904 20008000           750           dcw     0020H, 0080H    ; next time, power
2908 00C00000           751           dcl     0000c000H       ; position 1
290C 40004000           752           dcw     0040H, 0040H    ; next time, power
2910 00000000           753           dcl     00000000H       ; position 2
2914 6000C000           754           dcw     0060H, 00c0H    ; next time, power
2918 0080FFFF           755           dcl     0FFFF8000H      ; position 3
291C 80008000           756           dcw     0080H, 0080H    ; next time, power
                        757
2920 00080000           758           dcl     00000800H       ; position 4
2924 58008000           759           dcw     0058H, 0080H    ; next time, power
2928 00300000           760           dcl     00003000H       ; position 5
292C 7000FF00           761           dcw     0070H, 00ffH    ; next time, power
2930 00000000           762           dcl     00000000H       ; position 6
2934 9000F000           763           dcw     0090H, 00f0H    ; next time, power
2938 00000000           764           dcl     00000000H       ; position 7
293C 9100F000           765           dcw     0091H, 00f0H    ; next time, power
                        766
                        767
2940                    768           END

ASSEMBLY COMPLETED,  NO ERROR(S) FOUND.
```

270061-B1

intel®

APPLICATION NOTE

AP-275

November 1986

An FFT Algorithm For MCS®-96 Products Including Supporting Routines and Examples

IRA HORDEN
MCO APPLICATIONS ENGINEER

Order Number: 270189-001

1.0 INTRODUCTION

Intel's 8096 is a 16-bit microcontroller with processing power sufficient to perform many tasks which were previously done by microprocessors or special building block computers. A new field of applications is opened by having this much power available on a single chip controller.

The 8096 can be used to increase the performance of existing designs based on 8051s or similar 8-bit controllers. In addition, it can be used for Digital Signal Processing (DSP) applications, as well as matrix manipulations and other processing oriented tasks. One of the tasks that can be performed is the calculation of a Fast Fourier Transform (FFT). The algorithm used is similar to that in many DSP and matrix manipulation applications, so while it is directly applicable to a specific set of applications, it is indirectly applicable to many more.

FFTs are most often used in determining what frequencies are present in an analog signal. By providing a tool to identify specific waveforms by their frequency components, FFTs can be used to compare signals to one another or to set patterns. This type of procedure is used in speech detection and engine knock sensors. FFTs also have uses in vision systems where they identify objects by comparing their outlines, and in radar units to detect the dopler shift created by moving objects.

This application note discusses how FFTs can be calculated using Intel's MCS®-96 microcontrollers. A review of fourier analysis is presented, along with the specific code required for a 64 point real FFT. Throughout this application note, it is assumed that the reader has a working knowledge of the 8096. For those without this background the following two publications will be helpful:

1986 Microcontroller Handbook

Using the 8096, AP-248

These books are listed in the bibliography, along with other good sources of information on the MCS-96 product family and on Fast Fourier Transforms.

2.0 PROGRAM OVERVIEW

This application note contains program modules which are combined to create a program which performs an FFT on an analog signal sampled by the on-board ADC (Analog to Digital Converter) of the 8097. The results of the FFT are then provided over the serial channel to a printer or terminal which displays the results. In the applications listed in the previous section, the data from this FFT program would be used directly by another program instead of being plotted. However, the plotted results are used here to provide an example of what the FFT does. There are four program modules discussed in this application note:

FFTRUN - Runs a 64 point FFT on its data buffer. It produces 32 14-bit complex output values and 32 14-bit output magnitudes. A fast square root routine and log conversion routine are included.

A2DCON - Fills one of two buffers with analog values at a set sample rate. The sample time can be as fast as 50 microseconds using 8x9xBH components.

PLOTSP - Plots the contents of a buffer to a serially connected printer. Routines are provided for console out and hexadecimal to decimal conversion and printing.

FTMAIN - The main module which controls the other modules.

Each of the modules will be described separately. In order to better understand how the programs work together, a brief tutorial on FFTs will be presented first, followed by descriptions of the programs in the order listed above.

The final program uses 64 real data points, taken from either a table or analog input 1. Each of the data points is a 16-bit signed number. The processing takes 12.5 milliseconds when internal RAM is used as the data space. If external RAM is used, 14 milliseconds are required. Larger FFTs can be performed by slightly modifying the programs. A 256-point FFT would take approximately 65 milliseconds, and a 1024-point version would require about 300 milliseconds.

In the program presented, the analog sampling time is set for 1 sample every 100 microseconds, providing the 64 samples in 6.4 milliseconds. The sampling time can be reduced to around 60 microseconds per point by changing a variable, and less than 50 microseconds by using the 8x9xBH series of parts, since they have a 22 microsecond A to D conversion time.

The programs are set up to be run in a sequence instead of concurrently. This provides the fastest operation if the sampling speed were reduced to the minimum possible. For the fastest operation above about 80 microseconds a sample, the programs could be run concurrently, but this would require some minor modifications of the program. Figure 1 shows the timing of the program as presented.

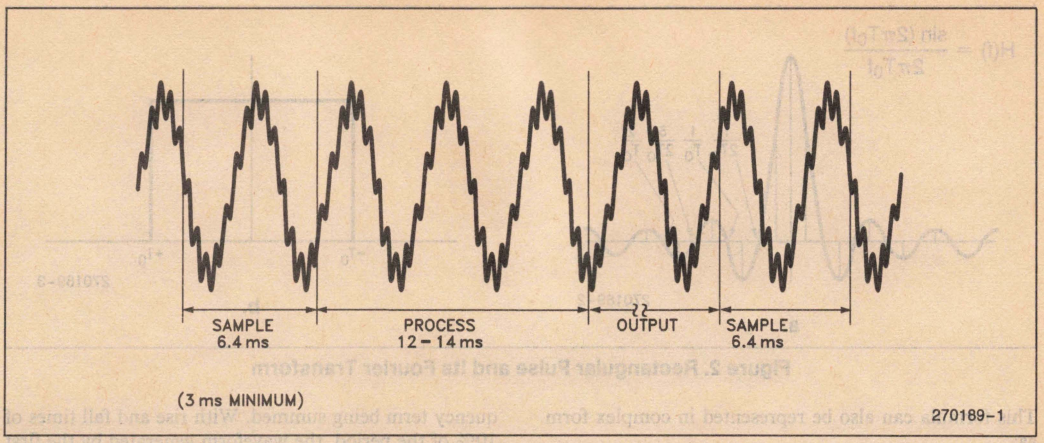


Figure 1. Timing of the FFT Program

These programs have run in the Intel Microcontroller Operation Application's Lab and produced the results presented in this application note. Since the programs have not undergone any further testing, we cannot guarantee them to be bug proof. We, therefore, recommend that they be thoroughly tested before being used for other than demonstration purposes.

3.0 FOURIER TRANSFORMS

A Fourier Transform is a useful analytical tool that is frequently ignored due to its mathematically oriented derivations. This is unfortunate, since Fourier transforms can be used without fully understanding the mathematics behind them. Of course, if one understands the theory behind these transforms, they become much more powerful.

The majority of this application note deals with how a Fast Fourier Transform (FFT) can be used for spectrum analysis. This procedure takes an input signal and separates it into its frequency components. One can almost treat the FFT as a black box, which has as its output, the frequency components and magnitudes of the input signal, much like a spectrum analyzer.

From a mathematical standpoint, Fourier Transforms change information in the time domain into the frequency domain. The theory behind the Fourier transform stems from Fourier analysis, also called frequency analysis.

There are many books on the topic of Fourier analysis, several of which are listed in the bibliography. In this application note, only the pertinent formulas and uses will be presented, not their derivations.

The main idea in Fourier analysis is that a function can be expressed as a summation of sinusoidal functions of different frequencies, phase angles, and magnitudes. This idea is represented by the Fourier Integral:

$$H(f) = \int_{-\infty}^{\infty} h(t)\, e^{-j2\pi ft} \qquad (1)$$

Where: H(f) is a function of frequency
h(t) is a function of time

Since

$$e^{-j\theta} = \cos\theta - j\sin\theta \qquad (2)$$

$$H(f) = \int_{-\infty}^{\infty} h(t)\,(\cos(2\pi ft) - j\sin(2\pi ft))\,dt \qquad (3)$$

Figure 2 shows a rectangular pulse and its Fourier transform. Note that the results in the frequency domain are continuous rather than discrete.

In a simplified case, the varying phase angles can be removed, and the integral changed to a summation, known as a Fourier Series. All periodic functions can be described in this way. This series, as shown below, can help provide a more graphical understanding of Fourier analysis.

$$y(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} [a_n \cos(2\pi nf_0t) + b_n \sin(2\pi nf_0t)] \qquad (4)$$

for $n = 1$ to ∞

Where $f_0 = \frac{1}{T_0}$, the fundamental frequency.

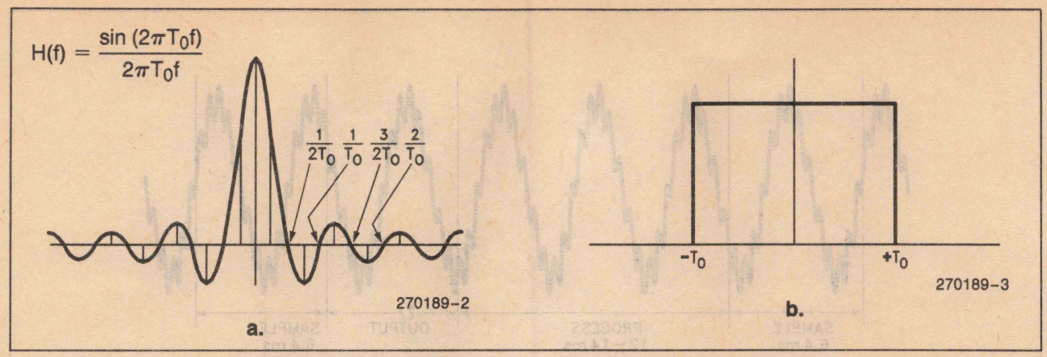


Figure 2. Rectangular Pulse and Its Fourier Transform

This formula can also be represented in complex form as:

$$\sum_{n=-\infty}^{\infty} \alpha n\, e^{j\pi n f_0 t} \qquad (5)$$

The Fourier series for a square wave is

$$\sum_{K=0}^{\infty} \frac{\sin((2k+1)\,2\pi f_0 t)}{(2k+1)} \qquad (6)$$

If these sinusoids are summed, a square wave will be formed. Figure 3 shows the graphical summation of the first 3 terms of the series. Since the higher frequencies contribute to the squareness of the waveform at the corners, it is reasonable to compare only the flatness of the top of the waveform. The sharpness or risetime of the waveform can be determined by the highest frequency term being summed. With rise and fall times of 10% of the period, the waveform generated by the first 3 terms is within 20% of ideal. At 7 terms it is within 10%, and at 20 terms it is within 5%. With a 5% risetime, it is within 20% of ideal after 5 terms, 10% after 13 terms and 5% after 32 terms. Figure 4 shows the resultant waveforms after the summation of 7, 15 and 30 terms.

Fourier analysis can be used on equation 4 to find the coefficients a_n and b_n. To make this process easier to use with a computer, a discrete form, rather than a continuous one, must be used. The discrete Fourier transform, shown in Equation 7, is a good approximation to the continuous version. The closeness of the approximation depends on several conditions which will be discussed later. The input to this transform is a set of N equally spaced samples of a waveform taken over a period of NT. The period NT is frequently referred to as the "Sampling Window".

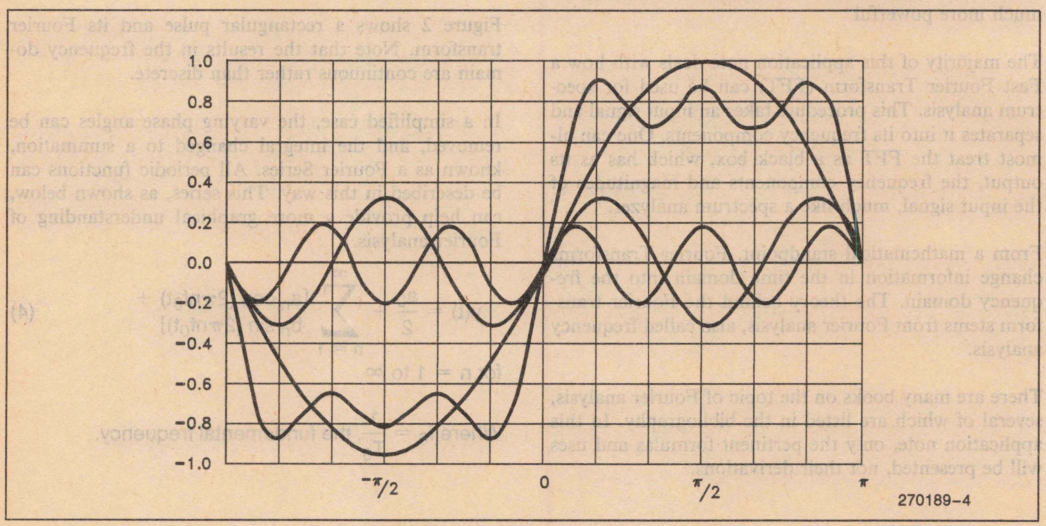
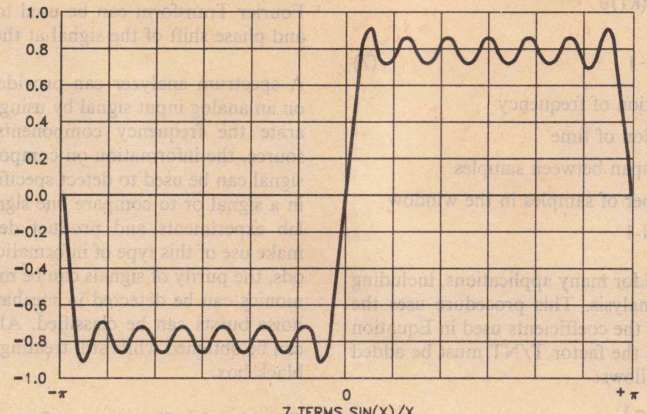
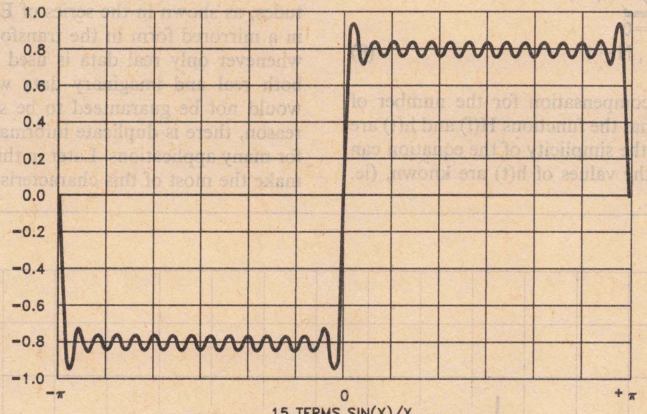


Figure 3. Graphical Summation of Sinewaves





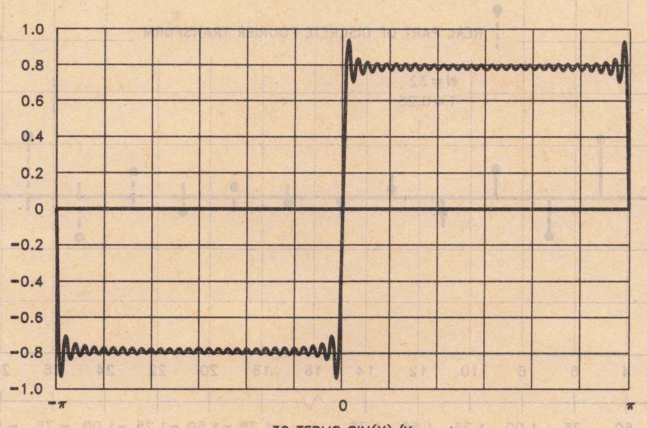


Figure 4. Square Wave from Sinusoids

$$H\left(\frac{n}{NT}\right) = \sum_{k=0}^{N-1} h(kT)e^{-j2\pi nk/N}$$

$$n = 0, 1, \ldots, N-1 \qquad (7)$$

Where: H(f) is a function of frequency

h(t) is a function of time

T is the time span between samples

N is the number of samples in the window

n =0,1,2 ... N-1

This transform is used for many applications, including Fourier Harmonic Analysis. This procedure uses the transform to calculate the coefficients used in Equation 5. In order to do this, the factor T/NT must be added to the transform as follows:

$$H\left(\frac{n}{NT}\right) = \frac{T}{(NT)} \sum_{k=0}^{N-1} h(kT)\, e^{-j2\pi nk/N}$$

$$n=0, 1, 2, 3, \ldots, N-1 \qquad (8)$$

The factor provides compensation for the number of samples taken. Note that the functions H(f) and h(t) are complex variables, so the simplicity of the equation can be misleading. Once the values of h(t) are known, (ie. the value of the input at the discrete times (t)), the Fourier Transform can be used to find the magnitude and phase shift of the signal at the frequencies (f).

A spectrum analyzer can provide similar information on an analog input signal by using analog filters to separate the frequency components. Regardless of its source, the information on component frequencies of a signal can be used to detect specific frequencies present in a signal or to compare one signal to another. Many lab experiments and product development tests can make use of this type of information. Using these methods, the purity of signals can be measured, specific harmonics can be detected in mechanical equipment, and noise bursts can be classified. All of this information can be obtained while still treating the FFT process as a black box.

Consider the discrete transform of a square wave as shown in Figure 5. Note that the component magnitudes, as shown in the series of Equation 6, are shown in a mirrored form in the transform. This will happen whenever only real data is used as the FFT input, if both real and imaginary data were used the output would not be guaranteed to be symmetrical. For this reason, there is duplicate information in the transform for many applications. Later in this section a method to make the most of this characteristic is discussed.

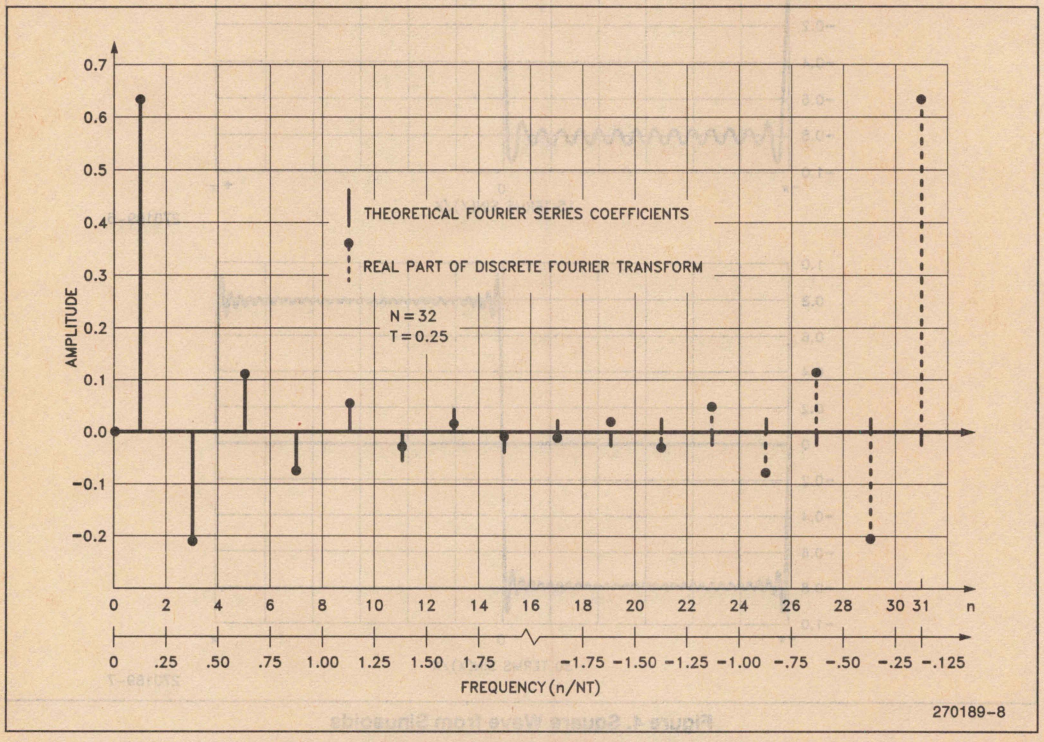


Figure 5. Discrete Transform of a Square Wave

If one looks at Equation 8, it can be seen that the calculation of a discrete Fourier transform requires N squared complex multiplications. If N is large, the calculation time can easily become unrealistic for real-time applications. For example, if a complex multiplication takes 40 microseconds, at N = 16, 10 milliseconds would be used for calculation, while at N = 128, over half a second would be needed. A Fast Fourier Transform is an algorithm which uses less multiplications, and is therefore faster. To calculate the actual time savings, it is first necessary to understand how a FFT works.

4.0 THE FFT ALGORITHM

The FFT algorithm makes use of the periodic nature of waveforms and some matrix algebra tricks to reduce the number of calculations needed for a transform. A more complete discussion of this is in Appendix A, however, the areas that need to be understood to follow the algorithm are presented here. This information need not be read if the reader's intent is to use the program and not to understand the mathematical process of the algorithm

To simplify notation the following substitutions are made in Equation 8.

$$W = e^{-j2\pi/N}$$

$$k = kT$$

$$n = \frac{n}{NT}$$

The resultant equation being

$$x(n) = \sum_{k=0}^{N-1} n(k)W^{nk} \qquad (9)$$

Expressed as a matrix operation

$$\begin{bmatrix} X(1) \\ X(2) \\ X(3) \\ \vdots \\ X(N\text{-}1) \end{bmatrix} = \begin{bmatrix} W^0 & W^0 & W^0 & \cdots & W^0 \\ W^0 & W^1 & W^2 & \cdots & W^N \\ W^0 & W^2 & W^4 & \cdots & W^{2N} \\ \vdots & \vdots & \vdots & & \vdots \\ W^0 & W^{(N-1)} & W^{2(N-1)} & \cdots & W^{(N-1)^2} \end{bmatrix} \begin{bmatrix} X_0(0) \\ X_0(1) \\ X_0(2) \\ \vdots \\ X_0(N-1) \end{bmatrix}$$

A brief review of matrix properties can be found in Appendix A. Because of the periodic nature of W the following is true:

$$W^{nk} \text{ MOD } N = W^{nk} \qquad (10)$$

$$= \cos(2\pi\, nk/N) - j \sin(2\pi nk/N)$$

$W^0 = 1$ therefore, if $nk \text{ MOD } N = 0$, $W^{nk} = 1$

This reduces the calculations as several of the W terms go to 1 and the highest power of W is N. All of W values are complex, so most of the operations will have to be complex operations. We will continue to use only the W, X(n) and X0(k) symbols to represent these complex quantities.

The FFT algorithm we will use requires that N be an integral power of 2. Other FFT algorithms do not have this restriction, but they are more complex to understand and develop. Additionally, for the relatively small values of N we are using this restriction should not provide much of a problem. We will define EXPONENT as log base 2 of N. Therefore,

$$N = 2^{EXPONENT}$$

The magic of the FFT, (as detailed in Appendix A), involves factoring the matrix into EXPONENT matrices, each of which has all zeros except for a 1 and a W^{nk} term in each row. When these matrices are multiplied together the result is the same as that of the multiplication indicated in Equation 9, except that the rows are interchanged and there are fewer non-trivial multiplications. To reorder the rows, and thus make the information useful, it is necessary to perform a procedure called "Bit Reversal".

This process requires that N first be converted to a binary number. The least significant bit (lsb) is swapped with the most significant bit (msb). Then the next lsb is swapped with the next msb, and so on until all bits have been swapped once. For N = 8, 3 bits are used, and the values for N and their bit reversals are shown below:

Number	Binary	Bit Reversal	Decimal BR
0	000	000	0
1	001	100	4
2	010	010	2
3	011	110	6
4	100	001	1
5	101	101	5
6	110	011	3
7	111	111	7

Recall that the FFT of real data provides a mirrored image output, but the FFT algorithm can accept inputs with both real and imaginary components. Since the inputs for harmonic analysis provided by a single A to D are real, the FFT algorithm is doing a lot of calculations with one input term equal to zero. This is obviously not very efficient. More information for a given size transform can be obtained by using a few more tricks.

It is possible to perform the FFT of two real functions at the same time by using the imaginary input values to the FFT for the second real function. There is then a post processing performed on the FFT results which separate the FFTs of the two functions. Using a similar procedure one can perform a transform on 2N real samples using an N complex sample transform.

The procedure involves alternating the real sample values between the real and imaginary inputs to the FFT. If, as in our example, the input to the FFT is a 2 by 32 array containing the complex values for 32 inputs, the 64 real samples would be loaded into it as follows:

N	00 01 02 03 04 05 06 07 ..... 30 31
REAL	00 02 04 06 08 10 12 14 ..... 60 62
IMAGINARY	01 03 05 07 09 11 13 15 ..... 61 63

This procedure is referred to as a pre-weave. In order to derive the desired results, the FFT is run, and then a post-weave operation is performed. The formula for the post-weave is shown below:

$$X_r(n) = \left[\frac{R(n)}{2} + \frac{R(N-n)}{2}\right] + \cos\frac{\pi n}{N}\left[\frac{I(N)}{2} + \frac{I(N-n)}{2}\right] - \sin\frac{\pi n}{N}\left[\frac{R(n)}{2} - \frac{R(N-n)}{2}\right] \quad n = 0, 1, \ldots, N-1$$

$$X_i(n) = \left[\frac{I(n)}{2} - \frac{I(N-n)}{2}\right] - \sin\frac{\pi n}{N}\left[\frac{I(n)}{2} + \frac{I(N-n)}{2}\right] - \cos\frac{\pi n}{N}\left[\frac{R(n)}{2} - \frac{R(N-n)}{2}\right] \quad n = 0, 1, \ldots, N-1 \qquad (11)$$

Where R(n) is the real FFT output value
I(n) is the imaginary FFT output value
Xr(n) is the real post-weave output
Xi(n) is the imaginary post-weave output

Note that the output is now one-sided instead of mirrored around the center frequency as it is in Figure 5. The magnitude of the signal at each frequency is calculated by taking the square root of the sum of the squares. The magnitude can now be plotted against frequency, where the frequency steps are defined as:

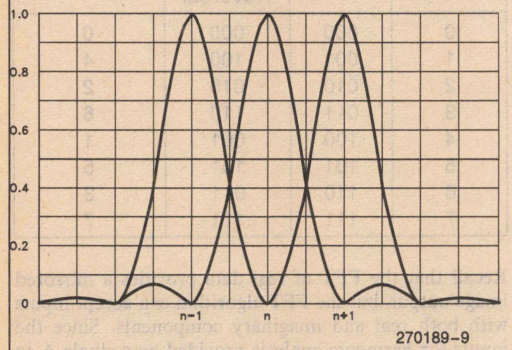
$$\frac{n}{NT} \quad n = 0, 1, 2, 3, \ldots, N-1$$

Where N is the number of complex samples (ie. 32 in this case) T is the time between samples

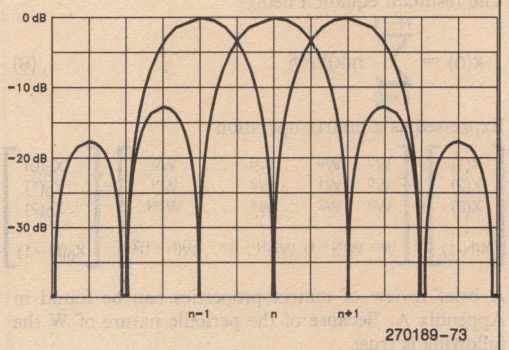
A value of zero on the frequency scale corresponds to the DC component of the waveform. Most signal analysis is done using Decibels (dB), the conversion is dB = 10 LOG (Magnitude squared). Decibels are not used as an absolute measure, instead signals are compared by the difference in decibels. If the ratio between two signals is 1:2 then there will be a 3 dB difference in their power.

5.0 USING THE FFT

There are several things to be aware of when using FFTs, but with the proper cautions, the FFT output can be used just like that of a spectrum analyzer. The



(a.) Relative Power of Windows (Side Lobes of Side Bins Removed for Clarity).



(b.) 10 Log Relative Power of Windows (Side Lobes of Side Bins Removed for Clarity).

Figure 6. Bin Windows

first precaution is that the FFT is a discrete approximation to a continuous Fourier Transform, so the output will seldom fit the theoretical values exactly, but it will be very close.

Since the programs in this application note generate a one-sided transform with N = 32, the frequency granularity is fairly course. Each of the frequency components output from the FFT is actually the sum of all energy within a narrow band centered on that frequency. This band of sensitivity is referred to as a "bin". The reported magnitude is the actual magnitude multiplied by the value of the bin window at the actual frequency. Figure 6 shows several bin windows. Note that these windows overlap, so that a frequency midway between the two center frequencies will be reported as energy split between both windows. Be careful not to confuse the *sampling window* NT with *bin windows* or with the *windowing function*.

Another area of caution is the relationship of the sampling window to the frequency of the waveform. For the best accuracy, the window should cover an exact multiple of the period of the waveform being analyzed. If it covers less than one period, the results will be invalid. Other variations from ideal will not produce invalid results, just additional noise in the output.

If the sampling window does not cover an exact multiple of all of the frequency components of a waveform, the FFT results will be noisy. The reason for this is the sharp edge that the FFT sees when the edges of the window cut off the input waveform. Figure 7 shows a waveform that is an exact multiple of the window and

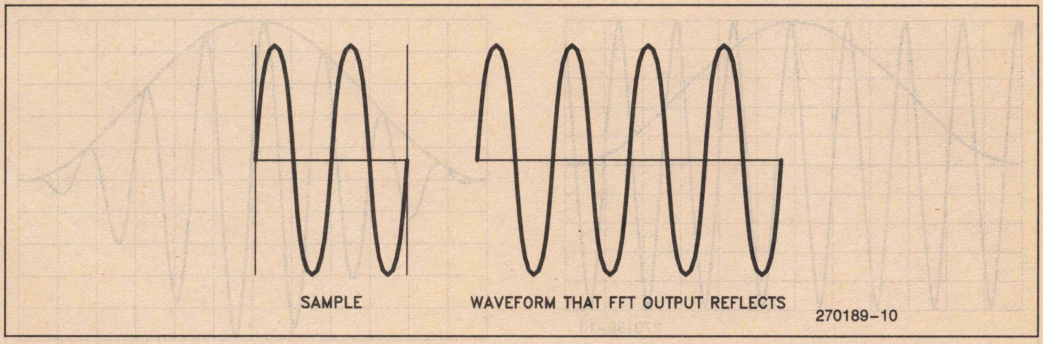


Figure 7. Waveform is a Multiple of the Window

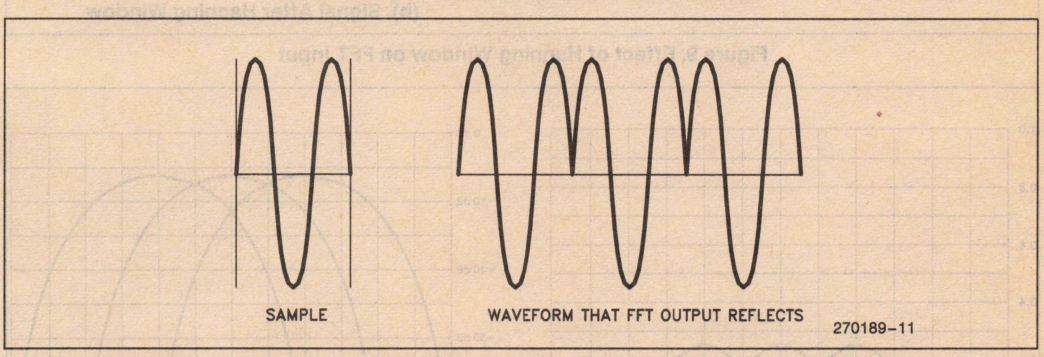
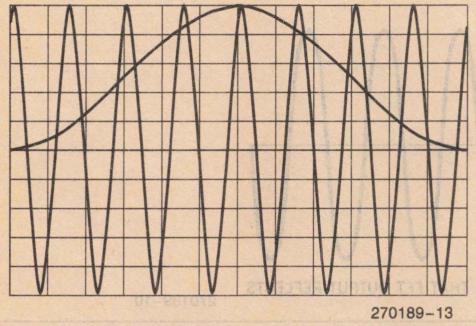


Figure 8. Waveform is Not a Multiple of the Window

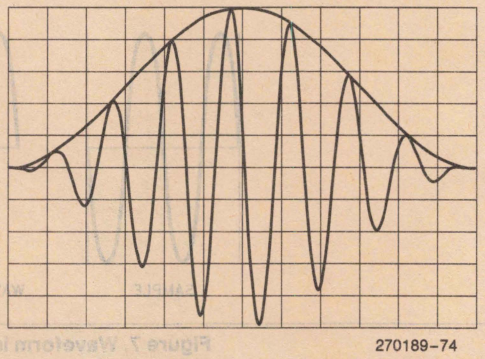
the periodic waveform that the FFT output reflects. In Figure 8, the waveform is not a multiple of the window and the waveform that the FFT output reflects has discontinuities. These discontinuities contribute to the noise in an FFT output. This noise is called "spectral leakage", or simply "leakage", since it is leakage between one frequency spectrum and another which is caused by digitization of an analog process.

To reduce this leakage, a process called windowing is used. In this procedure the input data is multiplied by specific values before being used in the FFT. The term "windowing" is used because these values act as a window through which the input data passes. If the input window goes smoothly to zero at both endpoints of the sampling window, there can be no discontinuities. Figure 9 shows a Hanning window and its effect on the input to an FFT. The Hanning window was named after its creator, Julius Von Hann, and is one of the most commonly used windows. More information on windowing and the types of windows can be found in the paper by Harris listed in the bibliography. As expected, the results of the FFT are changed because of the input windowing, but it is in a very predictable way.

Using the Hanning window results in bin windows which are wider and lower in magnitude than normal, as can be seen by comparing Figure 6 with Figure 10. For an input frequency which is equal to the center frequency of a window, the attenuation will be 6 dB on the center frequency. Since the bin windows are wider

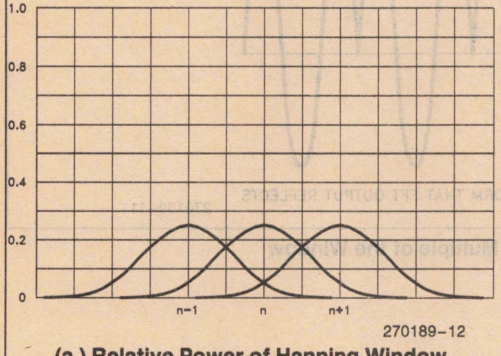


(a). Original Signal and Hanning Window

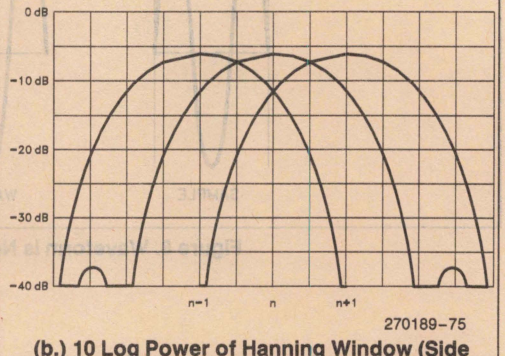


(b). Signal After Hanning Window

Figure 9. Effect of Hanning Window on FFT Input



(a.) Relative Power of Hanning Window



(b.) 10 Log Power of Hanning Window (Side Lobes of Side Bin Window Removed

Figure 10. Bin Windows after Using Hanning Input Window

than normal, the input frequency will also have energy which falls into the bins on either side of center. These side bins will show a reading of 6 dB below the center window. The disadvantage of this spreading is far less than the advantage of removing leakage from the FFT output.

A set of FFT output plots are included in the Appendix. These plots show the effect of windowing on various signals. There are examples of all of the cases described above. A brief discussion of the plots is also presented.

Applications which can make use of this frequency magnitude information include a wide range of signal processing and detection tasks. Many of these tasks use digital filtering and signature analysis to match signals to a standard. This technique has been applied to anti-knock sensors for automobile engines, object identification for vision systems, cardiac arrhythmia detectors, noise separation and many other applications. The ability to do this on a single-chip computer opens a door to new products which would have not been possible or cost effective previously.

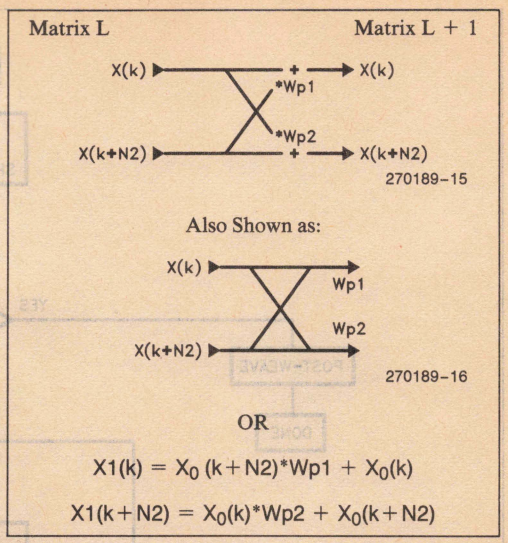
The next four sections of this application note cover the operation of the programs on a line by line basis. Section 6 shows an implementation of the FFT algorithm in BASIC. This code is used as a template to write the ASM96 code in Section 7. Sections 8, 9, and 10 cover the code sections which support the FFT module. After all of the code sections are discussed, an overview of how to use the program is presented in Section 11.

6.0 BASIC PROGRAM FOR FFTS

The algorithm for this FFT is shown in the flowchart in Figure 11 and the BASIC program in Listing 1. There are four sections to this program: initialization, pre-weaving, transform calculation, and post-weaving. The flowchart is generalized, however, the BASIC program has been optimized for assembly language conversion with 64 real samples.

On the flowchart, the initialization and pre-weaving sections are incorporated as "Read in Data". The data to be read includes the raw data as well as the size of the array and the scaling factor. The details for pre-weaving have been discussed earlier, and initialization varies from computer to computer. LOOP COUNT keeps track of which of the factored matrices are being multiplied. SHIFT is the shift count which is used to determine the power of W (as defined earlier) which will be used in the loop.

For each loop N calculations are performed in sets of two. Each calculation set is referred to as a butterfly and has the following form:



In general, the W factors are not the same. However, for the case of this FFT algorithm, Wp1 will always equal (−Wp2). This is because of the way in which "p" is calculated, and the fact that W(x) is a sinusoidal function.

The inner loop in the flowchart is performed N2 times. For LOOP = 1, N2 = N/2 and if INCNT = N2 then k = N2 and k + N2 = N, so the first loop is done and parameters LOOP, N2, and SHIFT are updated. For the first loop, all N/2 sets of calculations are performed contiguously. As LOOP increases, the number of contiguous calculations are cut in half, until LOOP = EXPONENT.

When LOOP = EXPONENT, N2 = 1, the butterfly is then performed on adjacent variables. Figure 12 shows the butterfly arrangement for a calculation where N = 8, so that EXPONENT = 3.

The BASIC program follows this flowchart, but operations have been grouped to make it easier to convert it to assembly language. Also not shown in the flowchart are several divide by 2 operations. There are five in the main section, one per loop. These provide the T/NT factor in equation 8 for N = 32 ($2^5 = 32$). There is also an extra divide by two in the post-weave section. It is required to prevent overflows when performing the 16-bit signed arithmetic in the ASM96 program. As a result of these operations, the input scale factor is $\pm 1 = \pm 32767$ and the output scaling is $\pm 1 = \pm 16384$. Note, the maximum input values are ± 0.99997.

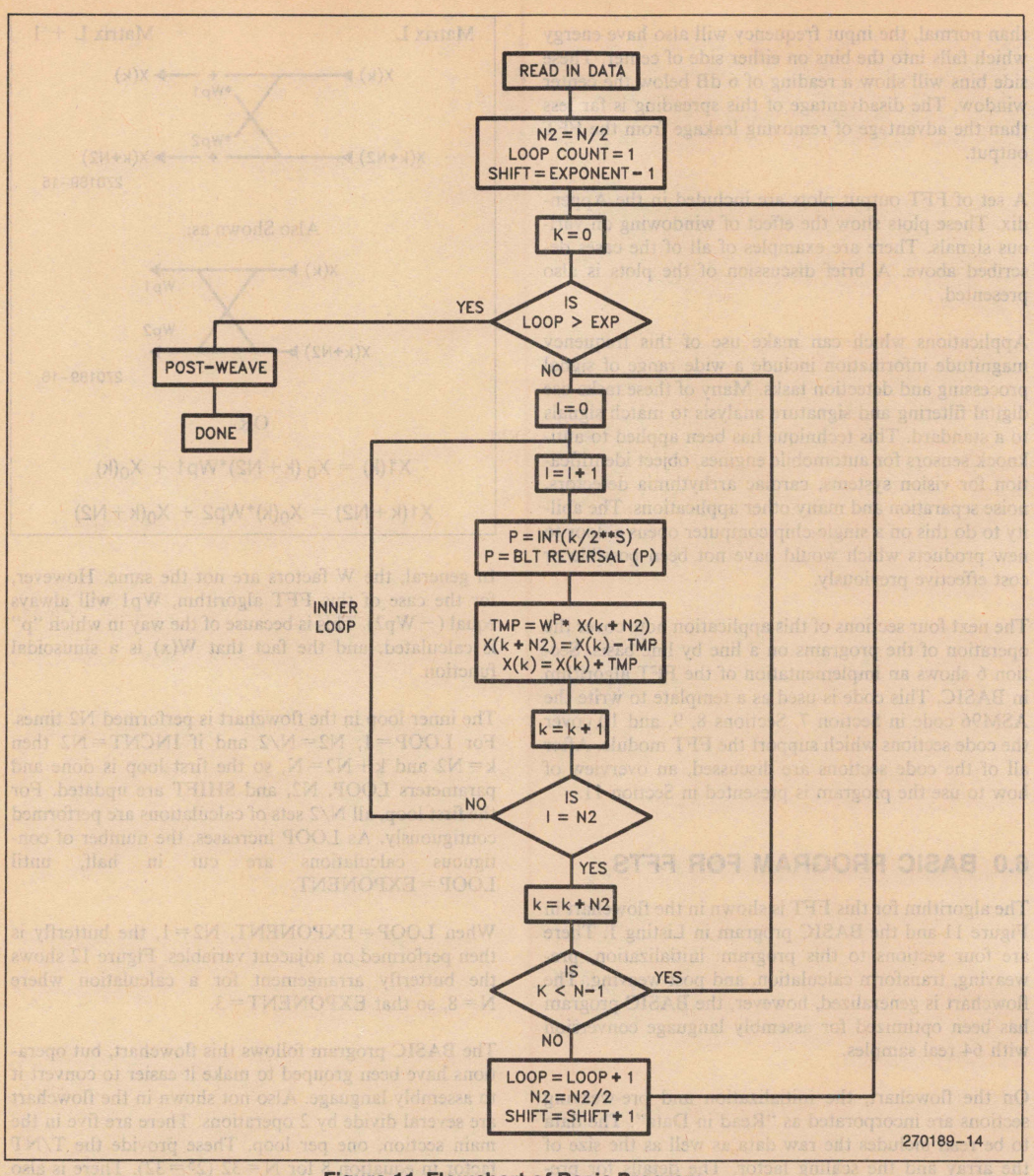


Figure 11. Flowchart of Basic Program

```
100 '  THIS IS FFT13, FEBRUARY 4, 1986
105 '
110     ' COPYRIGHT INTEL CORPORATION, 1985
115     ' BY IRA HORDEN, MCO APPLICATIONS
120 '
125 '   THIS PROGRAM PERFORMS A FAST FOURIER TRANSFORM ON 64 REAL DATA POINTS
130 ' USING A 2N-POINTS WITH AN N-POINT TRANSFORM ALGORITHM.  THE FIRST
135 ' SECTION OF THE PROGRAM PERFORMS A STANDARD TRANSFORM ON DATA THAT HAS
140 ' BEEN INTERLEAVED BETWEEN THE REAL AND IMAGINARY INPUT VALUES.  THE
145 ' RESULTS OF THAT TRANSFORM ARE THEN POST-PROCESSED IN THE SECOND SECTION
150 ' OF THE PROGRAM TO PROVIDE THE 32 OUTPUT BUCKETS.  THE OUTPUT VALUES ARE
155 ' MULTIPLIED BY "M" TO MAKE IT EASY TO COMPARE WITH THE ASM-96 PROGRAM
160 '
165 INPUT "NAME OF LIST FILE"; LST$
170 PRINT
175 OPEN LST$ FOR OUTPUT AS #1
180 '
200                                              ' SET UP VARIABLES FOR BASIC
210 DIM XR(32),XI(32),WR(32),WI(32),BR(32)
220 M=16383                                      ' M=MULT. FACTOR FOR SCALING
230 N=32 : N1=31 : N2=N/2                        ' N=NUMBER OF DATA POINTS
240 LOOP=1  : K=0 : EXPONENT=5 : SHIFT=EXPONENT-1       ' 2**E=N
250 PI=3.141592654# : TPN=2*PI/N : PIN=PI/N
260 '
270                                              ' READ IN CONSTANTS
280 FOR P=0 TO 31 : PN=P*TPN
290 WR(P)=COS(PN) : WI(P)=-SIN(PN) : READ BR(P)
300 NEXT P
310 '
320 FOR K=0 TO 31                               ' READ IN DATA
330 READ XR(K)  : READ XI(K)
350 NEXT K
360 '
400             ' INITIALIZATION OF LOOP
410 K=0
420 IF LOOP>EXPONENT THEN 700
430 INCNT=0
440             ' ACTUAL CALCULATIONS BEGIN HERE
445 '
450 INCNT=INCNT+1
460 P=BR(INT(K/(2^SHIFT)))
470 WRP=WR(P) : WIP=WI(P) : KN2=K+N2        ' WRP AND WIP ARE CONSTANTS BASED ON
480 TMPR= (WRP*XR(KN2) - WIP*XI(KN2))/2     ' SINES AND COSINES OF BIT REVERSED
490 TMPI= (WRP*XI(KN2) + WIP*XR(KN2))/2     ' VALUES OF K SHIFTED RIGHT S TIMES
500 TMPR1=XR(K)/2 : TMPI1=XI(K)/2
510 XR(K+N2) = TMPR1 - TMPR       ' TMPR, TMPI ARE THE REAL AND IMAGINARY
520 XI(K+N2) = TMPI1 - TMPI       ' RESULTS OF A COMPLEX MULTIPLICATION
530 XR(K) = TMPR1 + TMPR
540 XI(K) = TMPI1 + TMPI
550 '
560 K=K+1
570 IF INCNT<N2 THEN GOTO 450
580 K=K+N2                      ' SINCE THE ARRAY IS PROCESSED 2 POINTS AT A TIME,
590 IF K<N1 THEN GOTO 430       ' ONLY N/2 LOOPS NEED TO BE MADE.  ON EACH PASS,
600 LOOP=LOOP+1 : N2=N2/2       ' THE VALUE OF N2 CHANGES AND SMALLER CONSECUTIVE
605 SHIFT=SHIFT-1               ' SECTIONS ARE PROCESSED.
610 GOTO 400
620 '
690 '
691 '
692 '
693 '
```

270189-17

Listing 1—BASIC FFT Program

```
694 '
695 '
696 '
697 '
700                                ' POST-PROCESSING AND REORDERING BEGIN HERE
710 '
720 FOR K = 0 TO 31
730 KPIN=K*PIN
740 XRBRK=XR(BR(K)) : XIBRK=XI(BR(K))    ' CONDENSED FOR EASE OF ASM PROGRAMMING
750 XRBRNK=XR(BR(N-K)) : XIBRNK=XI(BR(N-K))
760 TI = (XIBRK+XIBRNK)/2
770 TR = (XRBRK-XRBRNK)/2
780 XRT= (XRBRK+XRBRNK)/4
790 XIT= (XIBRK-XIBRNK)/4
800 OUTR= XRT + TI*COS(KPIN)/2 - TR*SIN(KPIN)/2
810 OUTI= XIT - TI*SIN(KPIN)/2 - TR*COS(KPIN)/2
820 '
830 MAGSQ = OUTR*OUTR+OUTI*OUTI     ' THE ASM-96 PROGRAM USES A TABLE LOOK-UP
840 MAG = SQR(MAGSQ)                ' ROUTINE TO CALCULATE SQUARE ROOTS
845 IF MAGSQ*M < .5 THEN DECIBEL=0 : GOTO 900
847 DBFACT=M/2/32767*M   ' M^2 / 64K
850 DECIBEL=10*LOG(MAGSQ*DBFACT)
860 DECIBEL=DECIBEL * .434294481#
900    GOTO 930
910 PRINT #1, USING "######  "; K,
920 PRINT #1, USING "\       \"; HEX$(M*OUTR), HEX$(M*OUTI), HEX$(M*MAG)
930 ' GOTO 950
942 PRINT #1, USING "##  "; K;
943 PRINT #1, USING "##.#####  "; OUTR,OUTI,MAG;
945 PRINT #1, USING "###.###  "; DECIBEL;
947 PRINT #1, USING "######   "; M*OUTR, M*OUTI, M*MAG
950 NEXT K
960 '
970 IF LST$<>"SCRN:" THEN PRINT #1, CHR$(12)
999 END
1000 END
1010              ' DATA FOR BR(P)  - BIT REVERSAL
1020 DATA 0,16,8,24,4,20,12,28,2,18,10,26,6,22,14,30
1030 DATA 1,17,9,25,5,21,13,29,3,19,11,27,7,23,15,31
1040              ' DATA FOR XR,XI
1050 DATA 2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2
1060 DATA 2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2
1070 DATA -2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2
1080 DATA -2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2
```

270189-18

Listing 1—BASIC FFT Program (Continued)

Lines 165-175 set up the file for printing the data, this can be SCRN:, LPT1:, or any other file.

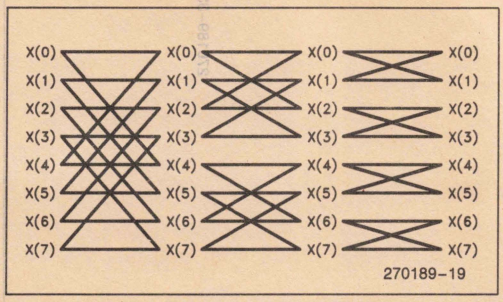


Figure 12. Butterflies with N = 8

Lines 200-310 set up the constants and calculate the WP terms which are stored in the matrices WR(p) and WI(p), for the real and imaginary component respectively.

Lines 320-350 read in the data, alternately placing it into the real and imaginary arrays. The data is scaled by 2 to make the data table simpler.

Lines 410-430 initialize the loop and test for completion.

Lines 450-620 perform the FFT algorithm. Note that all calculations are complex, with the suffixes "R" and "I" indicating real and imaginary components respectively.

The variables on line 470, TMPR1 and TMPI1 would normally not be used in a BASIC program as more than one operation can be performed on each line. However, indirect table lookups always use a separate line of assembly code, so separate lines have been used here.

Lines 700-810 perform the post-weave. This is not in the flowchart, but can be found in Equation 11. Once again, table look-ups are separated and additional variables are used for clarity. The variables BR(x) are the bit reversal values of x.

Line 830 calculates the magnitude of the harmonic components.

Lines 900-950 print the results of the calculations, with line 900 determining if the print-out should be in hex or decimal.

Lines 1000-1080 are the data for the bit reversal values and input datapoints. The input waveform is one cycle of a square-wave.

7.0 ASM96 PROGRAM FOR FFTS

The BASIC program just presented has been used as an outline for the ASM96 program shown in Listing 2. There are many advantages to using the BASIC program as a model, the main ones being debugging and testing. Since the BASIC program is so similar in program flow to the ASM96 program, it's possible to stop the ASM96 program at almost any point and verify that the results are correct.

```
MCS-96 MACRO ASSEMBLER    FFT_RUN                          02/18/86        PAGE    1

SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE: :F2:FFTRUN.A96
OBJECT FILE: :F2:FFTRUN.OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB

ERR LOC  OBJECT                 LINE        SOURCE STATEMENT
                                   1    $pagelength(50)
                                   2
                                   3    FFT_RUN  MODULE  STACKSIZE(6)
                                   4
                                   5    ; Intel Corporation, January 24, 1986
                                   6    ; by Ira Horden, MCO Applications
                                   7
                                   8
                                   9    ;       This module performs a fast fourier transform (FFT) on 64 real data
                                  10    ; points using a 2N-point algorithm.  The algorithm involves using a standard
                                  11    ; FFT procedure for 32 real and 32 imaginary numbers.  The real and imaginary
                                  12    ; arrays are filled alternately with real data points, and the output of the
                                  13    ; FFT is run through a post-processor.  The result is a one sided array with 32
                                  14    ; output buckets.  The post processing includes a table lookup algorithm for
                                  15    ; taking the square root of an unsigned 32-bit number.
                                  16
                                  17    ;     All of the calculations in the main FFT program are done using 16-bit
                                  18    ; signed integers.  The maximum value of any frequency component is therefore
                                  19    ; +/- 32K.  (Note that a square wave of +/-32K has a fundamental component
                                  20    ; greater than +/- 40K).  Wherever possible tables are used to increase the
                                  21    ; speed of math operations.  The complete transform, including obtaining the
                                  22    ; absolute magnitude of each frequency component, executes in 12
                                  23    ; milliseconds with internal variables, 14 ms with external.
                                  24
                                  25    ;     The program requires two 32-word input arrays, with the sample values
                                  26    ; alternated between the two.  These start at XREAL and XIMAG.  The resultant
                                  27    ; magnitude will be placed in a 32-word array at FFT_OUT.  These are all
                                  28    ; externally defined variables.  The external constant SCALE_FACTOR is used to
                                  29    ; divide the output when averaging will be used.  Since the program averages
                                  30    ; its output, it is necessary to clear the array based at FFT_OUT before
                                  31    ; calling FFT_CALC to start the program.
                                  32
                                  33    ;     The program was originally written in BASIC for testing purposes.  The
                                  34    ; comments include these BASIC statements to make it easier to follow the
                                  35    ; algorithm.
                                  36
                                  37    $EJECT
```

270189-33

Listing 2—ASM96 FFT Program

```
MCS-96 MACRO ASSEMBLER    FFT_RUN                                  02/18/86        PAGE    2

ERR LOC  OBJECT                LINE       SOURCE STATEMENT
                                 38
    0000                         39  RSEG
                                 40   EXTRN  port1, zero, error
                                 41
    0024                         42  OSEG at 24H
    0024                         43          TMPR:   dsl     1       ; Temporary register, Real
    0028                         44          TMPI:   dsl     1       ; Temporary register, Imaginary
    002C                         45          TMPR1:  dsl     1       ; Temporary register1, Real
    0030                         46          TMPI1:  dsl     1       ; Temporary register1, Imaginary
    0034                         47          XRTMP:  dsl     1       ; Temporary data register, Real
    0038                         48          XITMP:  dsl     1       ; Temporary data register, Imaginary
    003C                         49          XRRK:   dsl     1
    0040                         50          XRRNK:  dsl     1
    0044                         51          XIRK:   dsl     1
    0048                         52          XIRNK:  dsl     1
      003C                       53          diff    equ     xrrk    :long   ; Table difference for square root
      0040                       54          sqrt    equ     xrrnk   :long   ; Square root
      0040                       55          log     equ     xrrnk   :long   ; 10 Log magnitude^2
      0044                       56          nxtloc  equ     xirk    :long   ; Next location in table
                                 57
      003C                       58          WRP     equ     xrrk    :word   ; Multiplication factor, Real
      003E                       59          WIP     equ     xrrk+2  :word   ; Multiplication factor, Imaginary
      0040                       60          PWR     equ     xrrnk   :word
      0042                       61          IN_CNT  equ     xrrnk+2 :word
      0044                       62          NDIV2   equ     xirk    :word   ; n divided by 2  (0 < n < N) *2
                                 63
    004C                         64          KPTR:           dsw     1       ; K for counter *2 to index words
    004E                         65          KN2:            dsw     1       ; KPTR + NDIV2
    0050                         66          N_SUB_K:        dsw     1       ; N-K *2 to index words
    0052                         67          RK:             dsw     1       ; Bit reversed pointer of KPTR
    0054                         68          RNK:            dsw     1       ; Bit reversed pointer of N_SUB_K
    0056                         69          SHFT_CNT:       dsw     1
    0058                         70          LOOP_CNT:       dsb     1
      004E                       71          ptr     equ     kn2     :word   ; Pointer for square root table
    0000                         72  DSEG
                                 73
                                 74  EXTRN   FFT_MODE            ; FFT_MODE: mode for FFT input and graphing
                                 75  EXTRN   XREAL, XIMAG        ; XREAL, XIMAG: Base  addresses for 32 16-bit signed
                                 76                              ; entries for real and imaginary numbers respectively.
                                 77  EXTRN   FFT_OUT             ; FFT_OUT: Starting address for 32 word array
                                 78                              ; of magnitude information.
                                 79
    0000                         80          OUTR:   dsw     32      ; Real component of fft
    0040                         81          OUTI:   dsw     32      ; Imaginary component of waveform
                                 82   PUBLIC OUTR,OUTI
                                 83
                                 84  $EJECT
```

270189-34

```
MCS-96 MACRO ASSEMBLER    FFT_RUN                                  02/18/86          PAGE    3

ERR LOC  OBJECT                LINE       SOURCE STATEMENT
                                 85
    2280                         86  CSEG at 2280H
                                 87
                                 88   PUBLIC fft_calc        ; Starting point for FFT algorithm
                                 89
                                 90   EXTRN  scale_factor    ; Shift factor used to prevent overflow when averaging
                                 91                          ; fft outputs
                                 92
                                 93
                                 94  ;                                   ;;;;    START FOURIER CALCULATIONS
    2280                         95  FFT_CALC:                           ;;;;    400  ' INITIALIZATION OF LOOP
    2280 1100              E     96          clrb    error
    2282 B10100            E     97          ldb     port1,#00000001b                    ;**** Indication Only
                                 98
    2285 FC                      99          clrvt
    2286 B10158                 100          ldb     loop_cnt,#1
    2289 B10456                 101          ldb     shft_cnt,#4
    228C A1200044               102          ld      ndiv2,#32
                                103  ;                                   ;;;;    410 K=0
    2290                        104  OUT_LOOP:
    2290 950400            E    105          xorb    port1,#00000100B                    ;****  Indication Only
    2293 014C                   106          clr     kptr
                                107  ;                                   ;;;;    420 IF LOOP > EXP THEN 700
    2295 990558                 108          cmpb    loop_cnt, #5    ; 32=2^5
    2298 DA0220A3               109  !       bgt     UNWEAVE
                                110
                                111
    229C                        112  MID_LOOP:                           ;;;;    430 INCNT=0
    229C 0142                   113          clr     in_cnt
                                114
                                115                                     ;;;;    440  ' CALCULATIONS BEGIN HERE
    229E                        116  IN_LOOP:
    229E 65020042               117          add     in_cnt,#2          ;;;;    450 INCNT=INCNT+1
                                118  ;                                  ;;;;    460 P=BR(INT(K/(2^SHIFT)))
    22A2 A04C40                 119          ld      pwr,kptr
    22A5 085640                 120          shr     pwr,shft_cnt               ;; Calculate multiplication factors
    22A8 71FE40                 121          andb    pwr,#11111110B
    22AB A341003840             122          ld      pwr,brev[pwr]
                                123  ;                                   ;;;;    470 WRP=WR(P) : WIP=WI(P) : KN2=K+N2
    22B0 A34144393C             124  gw:     ld      wrp,wr[pwr]
    22B5 A34186393E             125          ld      wip,wi[pwr]
    22BA 44444C4E               126          add     kn2,kptr,ndiv2
                                127  $eject
```

270189-35

```
MCS-96 MACRO ASSEMBLER    FFT_RUN                              02/18/86          PAGE    4

ERR LOC  OBJECT                  LINE        SOURCE STATEMENT
                                  128           ;; Complex multiplication follows
                                  129
                                  130   ;                                 ;;;;    480 TMPR= (WRP*XR(KN2) - WIP*XI(KN2))/2
    22BE FE4F4F00003C24   E       131   gm:     mul     tmpr,wrp,xreal[kn2]
    22C5 FE4F4F00003E28   E       132           mul     tmpi,wip,ximag[kn2]
    22CC 682A26                   133           sub     tmpr+2,tmpi+2
                                  134   ;                                 ;;;;    490 TMPI= (WRP*XI(KN2) + WIP*XR(KN2))/2
    22CF FE4F4F00003C2C   E       135           mul     tmprl,wrp,ximag[kn2]
    22D6 FE4F4F00003E28   E       136           mul     tmpi,wip,xreal[kn2]
    22DD 642E2A                   137           add     tmpi+2,tmprl+2
                                  138
                                  139                   ;;  using the high byte only of a signed multiply
                                  140                   ;;  provides an effective divide by two
                                  141
    22E0 DC55                     142           BVT     ERR1    ; Branch on error in complex multiplications
                                  143
    22E2 A34D00002C       E       144           ld      tmprl,xreal[kptr]       ;;;;    500 TMPR1=XR(K)/2 :
    22E7 0A012C                   145           shra    tmprl,#1                ;;;;        TMPI1=XI(K)/2
    22EA A34D000030       E       146           ld      tmpil,ximag[kptr]
    22EF 0A0130                   147           shra    tmpil,#1
                                  148
                                  149   ;                                 ;;;;    510 XR(KN2) = TMPR1 - TMPR
    22F2 48262C34                 150   gr2:    sub     xrtmp,tmprl,tmpr+2
    22F6 C34F000034       E       151           st      xrtmp,xreal[kn2]
                                  152   ;                                 ;;;;    520 XI(KN2) = TMPI1 - TMPI
    22FB 482A3038                 153   gx2:    sub     xitmp,tmpil,tmpi+2
    22FF C34F000038       E       154           st      xitmp,ximag[kn2]
                                  155   ;                                 ;;;;    530 XR(K) = TMPR1 + TMPR
    2304 44262C34                 156           add     xrtmp,tmprl,tmpr+2
    2308 C34D000034       E       157           st      xrtmp,xreal[kptr]
                                  158   ;                                 ;;;;    540 XI(K) = TMPI1 + TMPI
    230D 442A3038                 159   gx:     add     xitmp,tmpil,tmpi+2
    2311 C34D000038       E       160           st      xitmp,ximag[kptr]
                                  161
    2316 DC23                     162           BVT     ERR2    ; Branch on error in complex additions
                                  163
                                  164   $eject
```

270189-36

Listing 2—ASM96 FFT Program (Continued)

```
MCS-96 MACRO ASSEMBLER    FFT_RUN                                  02/18/86        PAGE    5

ERR LOC  OBJECT                LINE       SOURCE STATEMENT
                                165  ;                              ;;;;    560 K=K+1
    2318 6502004C               166  ik:     add     kptr,#2
                                167
                                168  ;                              ;;;;    570 IF INCNT<N2 THEN  GOTO 450
    231C 884442                 169          cmp     in_cnt,ndiv2
    231F D602277B               170  !       blt     IN_LOOP
                                171
                                172  ;                              ;;;;    580 K=K+N2
    2323 64444C                 173          add     kptr,ndiv2
                                174  ;                              ;;;;    590 IF K<N1 THEN GOTO 430
    2326 893E004C               175          cmp     kptr,#62
    232A D602276E               176  !       blt     MID_LOOP
                                177
                                178                                 ;;;;    600 LOOP=LOOP+1 : N2=N2/2
    232E 1758                   179          incb    loop_cnt       ;;;;    605 SHIFT=SHIFT+1
    2330 0A0144                 180          shra    ndiv2,#1
    2333 1556                   181          decb    shft_cnt
                                182  ;                              ;;;;    610 GOTO 400
    2335 2759                   183          br      OUT_LOOP
                                184
                                185
    2337 B10100          E      186  ERR1:   ldb     error,#01      ; overflow error, 1st set of calculations
    233A F0                     187          ret
    233B B10200          E      188  ERR2:   ldb     error,#02      ; overflow error, 2nd set of calculations
    233E F0                     189          ret
                                190
                                191  $EJECT
```

270189-37

Listing 2—ASM96 FFT Program (Continued)

```
MCS-96 MACRO ASSEMBLER    FFT_RUN                                  02/18/86          PAGE    6

ERR LOC  OBJECT                LINE       SOURCE STATEMENT
                               192
                               193  ;                          ;;;;    700 ' POST-PROCESING AND REORDERING STARTS HERE
    233F                       194  UNWEAVE:
    233F B10200          E     195          ldb     portl,#00000010b                          ;****
                               196
                               197  ;                                   ;;;;    720 FOR K=0 TO 31
    2342 014C                  198          clr     kptr
    2344 A1400050              199          ld      n_sub_k,#64
    2348                       200  UN_LOOP:
                               201  ;                                   ;;;;    740 XIBRK=XI(BR(K)) : XRBRK=XR(BR(K)
    2348 A34D003852            202          ld      rk,brev[kptr]
    234D A35300003C      E     203          ld      xrrk,xreal[rk]
    2352 063C                  204          ext     xrrk
    2354 A353000044      E     205          ld      xirk,ximag[rk]
    2359 0644                  206          ext     xirk
                               207  ;                                   ;;;;    750 XIBRNK=XI(BR(N-K):XRBRNK=XR(BR(N-K)
    235B A351003854            208          ld      rnk,brev[n_sub_k]
    2360 A355000040      E     209          ld      xrrnk,xreal[rnk]
    2365 0640                  210          ext     xrrnk
    2367 A355000048      E     211          ld      xirnk,ximag[rnk]
    236C 0648                  212          ext     xirnk
                               213  ;                                   ;;;;    760 TI=(XIBRK + XIBRNK)/2
    236E 44484428              214  ar:      add     tmpi,xirk,xirnk
    2372 A04A2A                215          ld      tmpi+2,xirnk+2
    2375 A4462A                216          addc    tmpi+2,xirk+2
    2378 0E0128                217          shral   tmpi,#1         ; 16 bit result in tmpi
                               218
                               219                                     ;;;;    770 TR=(XRBRK - XRBRNK)/2
    237B 48403C24              220          sub     tmpr,xrrk,xrrnk
    237F A03E26                221          ld      tmpr+2,xrrk+2
    2382 A84226                222          subc    tmpr+2,xrrnk+2
    2385 0E0124                223          shral   tmpr,#1         ; 16 bit result in tmpr
                               224
                               225                                     ;;;;    780 XRT= (XRBRK + XRBRNK)/4
    2388 44403C34              226          add     xrtmp,xrrk,xrrnk
    238C A03E36                227          ld      xrtmp+2,xrrk+2
    238F A44236                228          addc    xrtmp+2,xrrnk+2
    2392 0D0E34                229          shll    xrtmp,#14        ; 32 bit result in xrtmp
                               230
                               231                                     ;;;;    790 XIT= (XIBRK-XIBRNK)/4
    2395 48484438              232          sub     xitmp,xirk,xirnk
    2399 A0463A                233          ld      xitmp+2,xirk+2
    239C A84A3A                234          subc    xitmp+2,xirnk+2
    239F 0D0E38                235          shll    xitmp,#14        ; 32 bit result in xitmp
                               236
                               237  $eject
```

270189-38

Listing 2—ASM96 FFT Program (Continued)

```
MCS-96 MACRO ASSEMBLER    FFT_RUN                                   02/18/86          PAGE    7

ERR LOC  OBJECT                  LINE       SOURCE STATEMENT
                                  238  ;       ;;;;;                    Multiply will provide effective divide by 2
                                  239
                                  240  ;                          ;;;;    800 OUTR= (XRT + TI*COSFN(K)/2 - TR*SINFN(K)/2)
                                  241
    23A2 FE4F4D4038242C           242  mr:       mul     tmprl,tmpr,sinfn[kptr]
    23A9 FE4F4DC2382830           243           mul     tmpil,tmpi,cosfn[kptr]
    23B0 643034                   244           add     xrtmp,tmpil
    23B3 A43236                   245           addc    xrtmp+2,tmpil+2
    23B6 682C34                   246           sub     xrtmp,tmprl
    23B9 A82E36                   247           subc    xrtmp+2,tmprl+2
    23BC C34D000036        R      248           st      xrtmp+2,outr[kptr]       ;; OUTR = Real Output Values
                                  249
                                  250
                                  251  ;                          ;;;;    810 OUTI= (XIT - TI*SINFN(K)/2 - TR*COSFN(K)/2)
                                  252
    23C1 FE4F4DC238242C           253  mi:       mul     tmprl,tmpr,cosfn[kptr]
    23C8 FE4F4D40382830           254           mul     tmpil,tmpi,sinfn[kptr]
    23CF 683038                   255           sub     xitmp,tmpil
    23D2 A8323A                   256           subc    xitmp+2,tmpil+2
    23D5 682C38                   257           sub     xitmp,tmprl
    23D8 682E3A                   258           sub     xitmp+2,tmprl+2
    23DB C34D40003A        R      259           st      xitmp+2,outi[kptr]       ;; OUTI = Imaginary Output values
                                  260
                                  261
                                  262                                   ;;;;    830 MAG =SQR(OUTR*OUTR + OUTI*OUTI)
                                  263
    23E0                          264  GET_MAG:                                 ;; Get Magnitude of Vector
    23E0 A03624                   265           ld      tmpr,xrtmp+2
    23E3 A03A28                   266           ld      tmpi,xitmp+2
                                  267
    23E6 FE6C2424                 268           mul     tmpr,tmpr                ; tmpr = tmpi**2 + tmpr**2
    23EA FE6C2828                 269           mul     tmpi,tmpi
    23EE 642824                   270           add     tmpr,tmpi
    23F1 A42A26                   271           addc    tmpr+2,tmpi+2
                                  272
    23F4 32004C            E      273           bbc     FFT_MODE,2,CALC_SQRT
                                  274
                                  275  $eject
```

270189-39

Listing 2—ASM96 FFT Program (Continued)

```
ERR LOC  OBJECT                LINE      SOURCE STATEMENT
                               276
                               277               ;;;;  ***  CALCULATE 10 log magnitude^2  ***
                               278   ; Output = 512*10*LOG(x)  x=1,2,3 ... 64K
                               279
    23F7                       280   CALC_LOG:
    23F7 0156                  281         clr     shft_cnt
    23F9 0F5624                282         norml   tmpr,shft_cnt   ; Normalize and get normalization factor
    23FC 990F56                283         cmpb    shft_cnt,#15
    23FF DA04                  284         jle     LOG_IN_RANGE    ;  Jump if SHIFT_CNT <= 15
                               285
    2401 0140                  286         clr     log
    2403 202C                  287         br      LOG_STORE
                               288
    2405                       289   LOG_IN_RANGE:
    2405 44565656              290         add     shft_cnt,shft_cnt,shft_cnt      ; Make shift_cnt a pointer
                               291
    2409 AC274E                292         ldbze   ptr,tmpr+3      ; Most significant byte is table pointer
    240C 444E4E4E              293         add     ptr,ptr,ptr     ;
    2410 65083A4E              294         add     ptr,# LOG_TABLE-256     ; ptr= Table + offset  (offset=tmpr+3)
                               295                                 ; Use -256 since tmpr+3 is always >= 128
    2414 A24F40                296         ld      log, [ptr]+
    2417 A24E44                297         ld      nxtloc, [ptr]           ;; Linear Interpolation
                               298
    241A 684044                299         sub     nxtloc,log      ; nxtloc = next log - log
                               300
    241D AC263C                301         ldbze   diff,tmpr+2     ; diff+1 = nxtloc * tmpr+2 / 256
    2420 6C443C                302         mulu    diff,nxtloc
                               303
    2423 0C083C                304         shrl    diff,#8         ; log = log + diff/256
    2426 643C40                305         add     log,diff
    2429 080540                306         shr     log,#5          ; 8192/32 * 20LOG(x) = 256 * 20LOG(x)
                               307
    242C A7570A3C40            308         addc    log,log_offset[shft_cnt]        ; add log of normalization factor
                               309
                               310               ;; Log (M*N) = Log M + Log N
                               311
    2431                       312   LOG_STORE:
    2431 080040           E   313         shr     log,#SCALE_FACTOR
    2434 A40040           E   314         addc    log,zero                ; Divide to prevent overflow during
    2437 674D000040       E   315         add     log,FFT_OUT[kptr]       ; averaging of outputs
    243C C34D000040       E   316         st      log,FFT_OUT[kptr]
                               317
    2441 2045                  318         BR      ENDL
                               319   $eject
```

270189-40

```
MCS-96 MACRO ASSEMBLER    FFT_RUN                                    02/18/86        PAGE    9

ERR LOC  OBJECT              LINE      SOURCE STATEMENT
                             320                        ;;;;  ***  CALCULATE SQUARE  ROOT  ***
    2443                     321   CALC_SQRT:
                             322
    2443 0156                323   clr     shft_cnt
    2445 0F5624              324           normL   tmpr,shft_cnt   ; Normalize and get normalization factor
                             325
    2448 D705                326           jne     SQRT_IN_RANGE   ;  Jump if tmpr > 0
    244A C04200          E   327           st      zero,sqrt+2
    244D 2029                328           br      SQRT_STORE
                             329
    244F                     330   SQRT_IN_RANGE:
    244F AC274E              331           ldbze   ptr,tmpr+3      ; Most significant byte is table pointer
    2452 444E4E4E            332           add     ptr,ptr,ptr
    2456 6508394E            333           add     ptr,# SQ_TABLE-256      ; ptr= Table + offset  (offset=tmpr+3)
                             334                                   ; Use -256 since tmpr+3 is always >= 128
    245A A24F40              335           ld      sqrt, [ptr]+
    245D A24E44              336           ld      nxtloc, [ptr]           ;; Linear Interpolation
                             337
    2460 684044              338           sub     nxtloc,sqrt     ; nxtloc =  sqrt - next sqrt
                             339
    2463 AC263C              340           ldbze   diff,tmpr+2     ; diff+1 = nxtloc * tmpr+2 / 256
    2466 6C443C              341           mulu    diff,nxtloc
                             342
    2469 AC3D3C              343           ldbze   diff,diff+1     ; sqrt = sqrt + delta   (diff < 0FFH)
    246C 643C40              344           add     sqrt,diff
                             345
    246F 44565656            346           add     shft_cnt,shft_cnt,shft_cnt
                             347
    2473 6F57C83940          348           mulu    sqrt,tab_sqr[shft_cnt]  ; divide by normalization factor
                             349
                             350                                   ;; mulu acts as divide since if tab2=0FFFFH
                             351                                   ;; sqrt would remain essentialy unchanged
    2478                     352   SQRT_STORE:
    2478 080042          E   353           shr     sqrt+2,#SCALE_FACTOR
    247B A40042          E   354           addc    sqrt+2,zero             ; Divide to prevent overflow during
    247E 674D000042      E   355           add     sqrt+2,FFT_OUT[kptr]    ; averaging of outputs
    2483 C34D000042      E   356           st      sqrt+2,FFT_OUT[kptr]
                             357
                             358   ;               ;;;;    ***  END  OF  LOOP  ***
                             359
                             360                                   ;;;;    950 NEXT K
    2488 6502004C            361   ENDL:   add     kptr,#2
    248C 69020050            362           sub     n_sub_k,#2
    2490 DF0226B4            363   !       bne     UN_LOOP
                             364
    2494 F0                  365           RET
                             366   $eject
```

270189-41

```
MCS-96 MACRO ASSEMBLER    FFT_RUN                                    02/18/86        PAGE   10

ERR LOC  OBJECT              LINE      SOURCE STATEMENT
                              367  ;$nolist
    3800                      368          CSEG  AT  3800H          ;;;;    Use 2k for tables
                              369
    3800                      370  BREV:   ; 2*bit reversal value
                              371
    3800 0000200010003000     372  DCW     2*0, 2*16, 2*8,  2*24, 2*4, 2*20, 2*12, 2*28
    3810 0400240014003400     373  DCW     2*2, 2*18, 2*10, 2*26, 2*6, 2*22, 2*14, 2*30
    3820 0200220012003200     374  DCW     2*1, 2*17, 2*9,  2*25, 2*5, 2*21, 2*13, 2*29
    3830 0600260016003600     375  DCW     2*3, 2*19, 2*11, 2*27, 2*7, 2*23, 2*15, 2*31
                              376
    3840                      377  SINFN:
    3840 00008C0CF9182825     378  DCW         0,   3212,   6393,   9512,  12539,  15446,  18204,  20787
    3850 825AF1626D6AE270     379  DCW     23170,  25329,  27245,  28898,  30273,  31356,  32137,  32609
    3860 FF7F617F897D7C7A     380  DCW     32767,  32609,  32137,  31356,  30273,  28898,  27245,  25329
    3870 825A33511C47563C     381  DCW     23170,  20787,  18204,  15446,  12539,   9512,   6393,   3212
    3880 000074F307E7D8DA     382  DCW         0,  -3212,  -6393,  -9512, -12539, -15446, -18204, -20787
    3890 7EA50F9D93951E8F     383  DCW    -23170, -25329, -27245, -28898, -30273, -31356, -32137, -32609
    38A0 01809F8077828485     384  DCW    -32767, -32609, -32137, -31356, -30273, -28898, -27245, -25329
    38B0 7EA5CDAEE4B8AAC3     385  DCW    -23170, -20787, -18204, -15446, -12539,  -9512,  -6393,  -3212
    38C0 0000                 386  DCW         0
                              387
    38C2                      388  COSFN:
    38C2 FF7F617F897D7C7A     389  DCW     32767,  32609,  32137,  31356,  30273,  28898,  27245,  25329
    38D2 825A33511C47563C     390  DCW     23170,  20787,  18204,  15446,  12539,   9512,   6393,   3212
    38E2 000074F307E7D8DA     391  DCW         0,  -3212,  -6393,  -9512, -12539, -15446, -18204, -20787
    38F2 7EA50F9D93951E8F     392  DCW    -23170, -25329, -27245, -28898, -30273, -31356, -32137, -32609
    3902 01809F8077828485     393  DCW    -32767, -32609, -32137, -31356, -30273, -28898, -27245, -25329
    3912 7EA5CDAEE4B8AAC3     394  DCW    -23170, -20787, -18204, -15446, -12539,  -9512,  -6393,  -3212
    3922 00008C0CF9182825     395  DCW         0,   3212,   6393,   9512,  12539,  15446,  18204,  20787
    3932 825AF1626D6AE270     396  DCW     23170,  25329,  27245,  28898,  30273,  31356,  32137,  32609
    3942 FF7F                 397  DCW     32767
                              398
    3944                      399  WR:             ;;;;    WR = COS(K*2PI/N)
    3944 FF7F897D41766D6A     400  DCW     32767,  32137,  30273,  27245,  23170,  18204,  12539,   6393
    3954 000007E705CFE4B8     401  DCW         0,  -6393, -12539, -18204, -23170, -27245, -30273, -32137
    3964 01807782BF899395     402  DCW    -32767, -32137, -30273, -27245, -23170, -18204, -12539,  -6393
    3974 0000F918FB301C47     403  DCW         0,   6393,  12539,  18204,  23170,  27245,  30273,  32137
    3984 FF7F                 404  DCW     32767
                              405
    3986                      406  WI:             ;;;;    WI = -SIN(K*2PI/N)
    3986 000007E705CFE4B8     407  DCW        -0,  -6393, -12539, -18204, -23170, -27245, -30273, -32137
    3996 01807782BF899395     408  DCW    -32767, -32137, -30273, -27245, -23170, -18204, -12539,  -6393
    39A6 0000F918FB301C47     409  DCW         0,   6393,  12539,  18204,  23170,  27245,  30273,  32137
    39B6 FF7F897D41766D6A     410  DCW     32767,  32137,  30273,  27245,  23170,  18204,  12539,   6393
    39C6 0000                 411  DCW         0
                              412  $eject
```

270189-42

```
ERR LOC  OBJECT              LINE      SOURCE STATEMENT
                             413
                             414
    39C8                     415  TAB_SQR:        ; 65535/(square root of 2**SHFT_CNT) ;  0<=SHFT_CNT<32
                             416
                             417    ;;    1       2       4       8       16      32      64      128
    39C8 FFFF04B50080825A    418  DCW     65535,  46340,  32768,  23170,  16384,  11585,  8192,   5793
                             419
                             420    ;;    256     512     1024    2048    4096    8192    16384   32768
    39D8 0010500B0008A805    421  DCW     4096,   2896,   2048,   1448,   1024,   724,    512,    362
                             422
                             423    ;;    65536,  131072, 262144, 524288, ...
    39E8 0001B50080005B00    424  DCW     256,    181,    128,    91,     64,     45,     32,     23
    39F8 10000B0008000600    425  DCW     16,     11,     8,      6,      4,      3,      2,      1
                             426
                             427
    3A08                     428  SQ_TABLE:       ; square root of n * 2**24  N=128, 129, 130 ... 255
                             429
    3A08 05B5BAB56EB621B7    430  DCW     46341, 46522, 46702, 46881, 47059, 47237, 47415, 47591
    3A18 97BA46BBF5BBA3BC    431  DCW     47767, 47942, 48117, 48291, 48465, 48637, 48809, 48981
    3A28 00C0AAC054C1FDC1    432  DCW     49152, 49322, 49492, 49661, 49830, 49998, 50166, 50332
    3A38 43C5E9C58EEC633C7   433  DCW     50499, 50665, 50830, 50995, 51159, 51323, 51486, 51649
    3A48 63CA04CBA6CB46CC    434  DCW     51811, 51972, 52134, 52294, 52454, 52614, 52773, 52932
    3A58 62CF00D09DD03AD1    435  DCW     53090, 53248, 53405, 53562, 53719, 53874, 54030, 54185
    3A68 44D4DED477D511D6    436  DCW     54340, 54494, 54647, 54801, 54954, 55106, 55258, 55410
    3A78 09D9A0D936DACCDA    437  DCW     55561, 55712, 55862, 56012, 56162, 56311, 56459, 56608
    3A88 B4DD47DEDBDE6EDF    438  DCW     56756, 56903, 57051, 57198, 57344, 57490, 57636, 57781
    3A98 46E2D7E267E3F7E3    439  DCW     57926, 58071, 58215, 58359, 58503, 58646, 58789, 58931
    3AA8 C1E64FE7DDE76AE8    440  DCW     59073, 59215, 59357, 59498, 59639, 59779, 59919, 60059
    3AB8 27EBB2EB3DECC7EC    441  DCW     60199, 60338, 60477, 60615, 60754, 60891, 61029, 61166
    3AC8 77EF00F08BF010F1    442  DCW     61303, 61440, 61576, 61712, 61848, 61984, 62119, 62254
    3AD8 B4F33BF4C1F446F5    443  DCW     62388, 62523, 62657, 62790, 62924, 63057, 63190, 63323
    3AE8 DFF763F8E7F86AF9    444  DCW     63455, 63587, 63719, 63850, 63982, 64113, 64243, 64374
    3AF8 F8FB7AFCFBFC7DFD    445  DCW     64504, 64634, 64763, 64893, 65022, 65151, 65280, 65408
                             446
                             447  $eject
```

270189-43

```
MCS-96 MACRO ASSEMBLER    FFT_RUN                                  02/18/86        PAGE   12

ERR LOC  OBJECT                  LINE        SOURCE STATEMENT
                                  448
    3B08                          449   LOG_TABLE:  ;   16384*10*LOG(n/128)     n=128,129,130 ... 256
                                  450
    3B08 00002A024F047006         451   DCW          0,   554,  1103,  1648,  2190,  2727,  3260,  3789
    3B18 DA10E312E914EA16         452   DCW       4314,  4835,  5353,  5866,  6376,  6883,  7386,  7885
    3B28 BD20A92292247826         453   DCW       8381,  8873,  9362,  9848, 10330, 10810, 11286, 11758
    3B38 C42F973166333335         454   DCW      12228, 12695, 13158, 13619, 14076, 14531, 14983, 15432
    3B48 063EC13F7A413043         455   DCW      15878, 16321, 16762, 17200, 17635, 18067, 18497, 18925
    3B58 954B3C4DDF4E8150         456   DCW      19349, 19772, 20191, 20609, 21024, 21436, 21846, 22254
    3B68 8458175AA85B365D         457   DCW      22660, 23063, 23464, 23862, 24259, 24653, 25045, 25435
    3B78 DE646066E0675D69         458   DCW      25822, 26208, 26592, 26973, 27353, 27730, 28106, 28479
    3B88 B370247294730275         459   DCW      28851, 29220, 29588, 29954, 30318, 30680, 31040, 31399
    3B98 0B7C6E7DCF7E2F80         460   DCW      31755, 32110, 32463, 32815, 33165, 33512, 33859, 34203
    3BA8 F28647889B89ED8A         461   DCW      34546, 34887, 35227, 35565, 35902, 36236, 36570, 36901
    3BB8 7091B892FF934595         462   DCW      37232, 37560, 37887, 38213, 38537, 38860, 39181, 39501
    3BC8 8B9BC89C049E3E9F         463   DCW      39819, 40136, 40452, 40766, 41079, 41390, 41700, 42009
    3BD8 4CA57EA6AFA7DEA8         464   DCW      42316, 42622, 42927, 43230, 43533, 43833, 44133, 44431
    3BE8 B9AEE0AF07B12CB2         465   DCW      44729, 45024, 45319, 45612, 45905, 46196, 46486, 46774
    3BF8 D6B7F4B811BA2DBB         466   DCW      47062, 47348, 47633, 47917, 48200, 48482, 48763, 49042
    3C08 A9C0                     467   DCW      49321
                                  468
    3C0A                          469   LOG_OFFSET:     ; 512*10*LOG(2**(15-n))    n= 0,1,2,3 ... 15
                                  470                     ; 512*10*LOG(0.5)          n= 16,17,18 ... 31
                                  471
    3C0A 4F5A4A54454E3F48         472   DCW      23119, 21578, 20037, 18495, 16954, 15413, 13871, 12330
    3C1A 252A20241A1E1518         473   DCW      10789,  9248,  7706,  6165,  4624,  3083,  1541,     0
                                  474
    3C2A                          475   END

ASSEMBLY COMPLETED,   NO ERROR(S) FOUND.
```

270189-44

Listing 2—ASM96 FFT Program (Continued)

The BASIC program is used as comments in the ASM96 program. Some of the variables in the ASM96 program have slightly different names than their counter-parts in the BASIC program. This was to make the comments fit into the ASM96 code. Highlights in this section of code are a table driven square root routine and log conversion routine which can easily be adapted for use by any program.

Both the square root routine and the log conversion routine use the 32-bit value in the variable TMPR. The square root routine calculates the square root of that value in the variable SQRT+2, a 16-bit variable. In this program, the square root value is averaged and stored in a table.

The log conversion routine divides the value in TMPR by 65536 (2^{16}) and uses table lookup to provide the common log. The result is a 16-bit number with the value 512 * 10 Log (TMPR/65536) stored in the variable LOG. This calculation is used to present the results of the FFT in decibels instead of magnitude. With an input of 63095, the output is 512*48 dB. The graph program, (Section 10), prints the output value of the plot as INPUT/512 dB.

The following descriptions of the ASM code point out some of the highlights and not-so-obvious coding:

Lines 1-104 initialize the code and declare variables. The input and output arrays of the program are declared external. Note that many of the registers are overlayable, use caution when implementing this routine with others with overlayable registers.

Lines 116–124 calculate the power of W to be used. Note that KPTR is always incremented by 2. The multiple right shift followed by the AND mask creates an even address and the indirect look to the BR (Bit Reversal) table quickly calculates the power PWR.

Lines 130-138 perform the complex multiplications. Since WIP and WRP range from −32767 to +32767, the multiplication is easy to handle. The automatic divide by two which occurs when using the upper word only of the 32-bit result is a feature in this case.

Lines 144-163 use right shifts for a fast divide, then add or subtract the desired variables and store them in the array. Note that the upper word of TMPR and TMPI is used, and the same array is used for both the input and output of the operations.

Lines 165-189 update the loop variables and then check for errors on the complex multiplications and additions. If there are no overflows at this time the data will run smoothly through the rest of the program.

Lines 200-212 load variables with values based on the bit reversed values of pointers.

Lines 214-236 perform additions and subtractions to prepare for the next set of formulas. Note that XITMP and XRTMP are 32-bit values.

Lines 240-260 perform multiplies and summations resulting in 32-bit variables. This saves a bit or two of accuracy. The upper words are then stored as the results.

Lines 263-272 generate the squared magnitude of the harmonic component as a 32-bit value.

Lines 278-310 calculate 10 Log (TMPR/65536). The 32-bit register TMPR is divided by 65536 so that the output range would be reasonable.

First, the number is normalized. (It is shifted left until a 1 is in the most significant bit, the number of shifts required is placed in SHFT_CNT.) If it had to be shifted more than 15 times the output is set to zero.

Next, the most significant BYTE is used as a reference for the look-up table, providing a 16-bit result. The next most significant BYTE is then used to perform linear interpolation between the referenced table value and the one above it. The interpolated value is added to the directly referenced one.

The 16-bit result of this table look-up and interpolation is then added to the Log of the normalization factor, which is also stored in a table. This table look-up approach works fast and only uses 290 bytes of table space.

Lines 321-357 calculate the square root of the 32-bit register TMPR using a table look-up approach.

First, the number is normalized. Next, the most significant BYTE is used as a reference for the look-up table, providing a 16-bit result. The next most significant BYTE is then used to perform linear interpolation between the referenced table value and the one above it. The interpolated value is added to the directly referenced one.

The 16-bit result of this table look-up and interpolation is then divided by the square root of the normalization factor, which is also stored in a table. This table look-up approach works fast and only uses 320 bytes of table space. The results are valid to near 14-bits, more than enough for the FFT algorithm.

Lines 352-360 average the magnitude value, if multiple passes are being performed, and then store the value in the array. The loop-counters are incremented and the process repeats itself.

This concludes the FFT routine. In order to use it, it must be called from a main program. The details for calling this routine are covered in the next section.

8.0 BACKGROUND CONTROL PROGRAM

The main routine is shown in Listing 3. It begins with declarations that can be used in almost any program. Note that these are similar, but not identical, to other 8096 include files that have been published. Comments on controlling the Analog to Digital converter routine follow the declarations.

```
MCS-96 MACRO ASSEMBLER    FFT_MAIN_APNOTE                          02/18/86        PAGE    1

SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE: :F2:FTMAIN.A96
OBJECT FILE: :F2:FTMAIN.OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB

ERR LOC  OBJECT                 LINE      SOURCE STATEMENT
                                  1   $pagelength(50)
                                  2
                                  3   FFT_MAIN_APNOTE MODULE MAIN, STACKSIZE(6)
                                  4
                                  5   ; Intel Corporation, January 24, 1986
                                  6   ; by Ira Horden, MCO Applications
                                  7
                                  8
                                  9   ;    This program performs an FFT on real data and plots it on a printer.
                                 10   ; It uses the program modules A2DCON, PLOTSP, and FFTRUN.  The adjustable
                                 11   ; parameters of each of the programs are set by this main module.
                                 12
                                 13
                                 14   $INCLUDE (:F0:DEMO96.INC)       ; Include SFR definitions
                               =1 15   ;$nolist  ;  Turn listing off for include file
                               =1 16
                               =1 17   ;*******************************************************************************
                               =1 18   ;
                               =1 19   ;               Copyright 1985, Intel Corporation
                               =1 20   ;               October 28,1985
                               =1 21   ;               by Ira Horden, MCO Applications
                               =1 22   ;
                               =1 23   ; DEMO96.INC - DEFINITION OF SYMBOLIC NAMES FOR THE I/O REGISTERS OF THE 8096
                               =1 24   ;
                               =1 25   ;*******************************************************************************
                               =1 26   ;
0000                           =1 27   ZERO            EQU   00h:WORD      ; R/W   Zero Register
0002                           =1 28   AD_COMMAND      EQU   02H:BYTE      ;   W   A to D command register
0002                           =1 29   AD_RESULT_LO    EQU   02H:BYTE      ; R     Low byte of result and channel
0003                           =1 30   AD_RESULT_HI    EQU   03H:BYTE      ; R     High byte of result
0003                           =1 31   HSI_MODE        EQU   03H:BYTE      ;   W   Controls HSI transition detector
0004                           =1 32   HSO_TIME        EQU   04H:WORD      ;   W   HSI time tag
0004                           =1 33   HSI_TIME        EQU   04H:WORD      ; R     HSO time tag
0006                           =1 34   HSO_COMMAND     EQU   06H:BYTE      ;   W   HSO command tag
0006                           =1 35   HSI_STATUS      EQU   06H:BYTE      ; R     HSI status register (reads fifo)
0007                           =1 36   SBUF            EQU   07H:BYTE      ; R/W   Serial port buffer
0008                           =1 37   INT_MASK        EQU   08H:BYTE      ; R/W   Interrupt mask register
0009                           =1 38   INT_PENDING     EQU   09H:BYTE      ; R/W   Interrupt pending register
0011                           =1 39   SPCON           EQU   11H:BYTE      ;   W   Serial port control register
0011                           =1 40   SPSTAT          EQU   11H:BYTE      ; R     Serial port status register
000A                           =1 41   WATCHDOG        EQU   0AH:BYTE      ;   W   Watchdog timer
```

270189-45

Listing 3—Main Routine

```
MCS-96 MACRO ASSEMBLER    FFT_MAIN_APNOTE                                  02/18/86          PAGE    2

ERR LOC  OBJECT                LINE      SOURCE STATEMENT
     000A                =1    42   TIMER1          EQU   0AH:WORD      ; R     Timer1 register
     000C                =1    43   TIMER2          EQU   0CH:WORD      ; R     Timer2 register
     000E                =1    44   PORT0           EQU   0EH:BYTE      ; R     I/O port 0
     000E                =1    45   BAUD_REG        EQU   0EH:BYTE      ;   W   Baud rate register
     000F                =1    46   PORT1           EQU   0FH:BYTE      ; R/W   I/O port 1
     0010                =1    47   PORT2           EQU   10H:BYTE      ; R/W   I/O port 2
     0015                =1    48   IOC0            EQU   15H:BYTE      ;   W   I/O control register 0
     0015                =1    49   IOS0            EQU   15H:BYTE      ; R     I/O status register 0
     0016                =1    50   IOC1            EQU   16H:BYTE      ;   W   I/O control register 1
     0016                =1    51   IOS1            EQU   16H:BYTE      ; R     I/O status register 1
     0017                =1    52   PWM_CONTROL     EQU   17H:BYTE      ;   W   PWM control register
     0018                =1    53   SP              EQU   18H:WORD      ; R/W   System stack pointer
                         =1    54
     000D                =1    55   CR              EQU   0DH
     000A                =1    56   LF              EQU   0AH
                         =1    57
                         =1    58   PUBLIC ZERO,AD_COMMAND,AD_RESULT_LO,AD_RESULT_HI,HSI_MODE,HSO_TIME,HSI_TIME
                         =1    59   PUBLIC HSO_COMMAND
                         =1    60   PUBLIC HSI_STATUS,SBUF,INT_MASK,INT_PENDING,WATCHDOG,TIMER1,TIMER2
                         =1    61   PUBLIC BAUD_REG, PORT0, PORT1, PORT2,SPSTAT,SPCON,IOC0,IOC1,IOS0,IOS1
                         =1    62   PUBLIC PWM_CONTROL,SP,CR,LF
                         =1    63
     001C                =1    64   RSEG at 1CH
                         =1    65
     001C                =1    66           AX:     DSW     1           ; Temp registers used in conformance
     001E                =1    67           DX:     DSW     1           ; with PLM-96(tm) conventions.
     0020                =1    68           BX:     DSW     1
     0022                =1    69           CX:     DSW     1
                         =1    70
       001C              =1    71           AL      EQU     AX      :BYTE
       001D              =1    72           AH      EQU     (AX+1)  :BYTE
       0020              =1    73           BL      EQU     BX      :BYTE
                         =1    74
                         =1    75    public ax, bx, cx, dx, al, ah, bl
                         =1    76
                         =1    77   $list    ;  Turn listing back on
                         =1    78            ;  End of include file
                               79
                               80   ; A2D UTILITY COMMANDS/RESPONSES FOR "CONTROL_A2D" ____________________________
                               81
       0007                    82   busy            equ     7
       0010                    83   con_b0          equ     00010000b;convert to BUFF0
       0028                    84   dump_b0_p_s     equ     00101000b; download  BUFF0  as PAIRED SIGNED data
                               85   ;_____________________________________________________________________________
                               86
       0001                    87   AVR_NUM         equ     1       ; Number of times to average the waveform
                               88                                   ;    AVR_NUM < 256
```

270189-46

```
MCS-96 MACRO ASSEMBLER    FFT_MAIN_APNOTE                          02/18/86          PAGE    3

ERR LOC  OBJECT                LINE       SOURCE STATEMENT
                                 89
      0000                       90  SCALE_FACTOR    equ     0       ; Number of rights shifts performed on
                                 91                                  ; output of FFT.  Used to prevent overflow
                                 92                                  ; on summation
                                 93
      0100                       94  PLOT_RES        equ     256             ; Number of input units per plot unit
      0080                       95  PLOT_RES_2      equ     plot_res/2
      9100                       96  PLOT_MAX        equ     plot_res*145    ; 145 chrs/row
                                 97
                                 98   PUBLIC scale_factor, plot_res, plot_res_2, plot_max
                                 99
                                100
      0024                       101 OSEG at 24H     ; common oseg area
                                102
      0024                       103         tmpreal:        dsl     1
      0028                       104         tmpimag:        dsl     1
      002C                       105         wndptr:         dsw     1
      002E                       106         varptr:         dsw     1
                                107
      0000                       108 RSEG
      0000                       109         fft_mode:       dsb     1
      0001                       110         error:          dsb     1
      0002                       111         avr_cnt:        dsb     1
                                112   PUBLIC error, fft_mode
                                113
                                114         EXTRN   sample_period, control_a2d
                                115
                                116
      0080                       117 DSEG at 80h
      0080                       118         XREAL:                          ; For FFT routine
      0080                       119         DEST_BUFF_BASE: DSW     64      ; For A2D routine
        00C0                     120         XIMAG equ       XREAL+64        ; For FFT routine
                                121
                                122   PUBLIC DEST_BUFF_BASE, XREAL, XIMAG
                                123
                                124
      0200                       125 DSEG AT 200H
                                126
      0200                       127         PLOT_IN:
      0200                       128         FFT_OUT:        DSW     32      ; For FFT routine
      0240                       129         BUFF0_BASE:     DSW     64      ; For A2D routine
      02C0                       130         BUFF1_BASE:     DSW     64      ; For A2D routine
                                131
                                132         PUBLIC  BUFF0_BASE, BUFF1_BASE, FFT_OUT, PLOT_IN
                                133 $eject
```

270189-47

Listing 3—Main Routine (Continued)

```
ERR LOC  OBJECT              LINE      SOURCE STATEMENT
                             134
    2080                     135  CSEG AT 2080H
                             136
                             137          EXTRN   INIT_OUTPUT, DRAW_GRAPH, CON_OUT      ; For Plot Routine
                             138          EXTRN   FFT_CALC                            ; For FFT routine
                             139          EXTRN   A2D_BUFF_UTIL                       ; For A2D routine
                             140
    2080 A1000018        R   141          LD      SP,#STACK
    2084 A30100301C          142          LD      AX,3000H
    2089                     143  SBE_WAIT:
    2089 E01CFD              144          djnz    al,sbe_wait     ; WAIT FOR SBE TO CLEAR SERIAL PORT INTERRUPTS
    208C E01DFA              145          djnz    ah,sbe_wait
                             146
    208F EF0000          E   147  BEGIN:  CALL    INIT_OUTPUT     ; Initialize serial port
                             148
    2092                     149  NEW_TRANSFORM_SET:
    2092 B10000          R   150          ldb     fft_mode,#0000B         ;  Bit 0 - Real data / Tabled data#
                             151                                          ;  Bit 1 - Windowed / Unwindowed#
                             152                                          ;  Bit 2 - 10log Mag^2 / Magnitude#
                             153                                          ;  Bit 3 - 256*db plot / Normal Plot#
    2095 B10102          R   154          ldb     avr_cnt,#avr_num
    2098 0120                155          clr     bx
    209A C321000200          156  CLRRAM: st      zero,fft_out[bx]        ; clear fft magnitude array
    209F 65020020            157          add     bx,#2
    20A3 89400020            158          cmp     bx,#64
    20A7 DEF1                159          blt     CLRRAM
                             160
    20A9 300004          R   161  C_load: bbc     fft_mode,0,do_tab       ; Branch if real data is not used
    20AC 2819                162          CALL    LOAD_DATA
    20AE 2002                163          br      C_win
                             164
    20B0 282F                165  do_tab: CALL    TABLE_LOAD
                             166
    20B2 310002          R   167  C_win:  bbc     fft_mode,1,calc         ; Branch if windowing is not used
    20B5 28CB                168          CALL    DO_WINDOW
                             169
    20B7 EF0000          E   170  CALC:   CALL    FFT_CALC
    20BA 980001          R   171  errtrp: cmpb    error,zero
    20BD D7FB                172          jne     errtrp
                             173
    20BF E00205          R   174          DJNZ    avr_cnt, LOAD_DATA      ; repeat for AVR_NUM counts
                             175
    20C2 EF0000          E   176          CALL    DRAW_GRAPH
                             177
    20C5 27CB                178          BR      NEW_TRANSFORM_SET
                             179  $eject
```

270189-48

```
MCS-96 MACRO ASSEMBLER    FFT_MAIN_APNOTE                    02/18/86        PAGE    5

ERR LOC  OBJECT                  LINE      SOURCE STATEMENT
                                  180  ;-------------------------------------------------------------------------
    20C7                          181  LOAD_DATA:                    ;;;;    LOAD DATA INTO RAM
                                  182
    20C7 B1000F                   183          ldb     port1,#00                       ;**** FOR INDICATION ONLY
                                  184
    20CA                          185  SET_A2D:
    20CA B11000              E    186          ldb     control_a2d,#con_b0     ; Set converter for buffer0
    20CD 910100              E    187          orb     control_a2d,#01         ; Convert channel 1
    20D0 A1320000            E    188          ld      sample_period,#50       ; 100 us sample period
                                  189
    20D4 EF0000              E    190          CALL    a2d_buff_util           ; Start the conversion process
    20D7 3F00FD              E    191          jbs     control_a2d,busy,$      ; wait for all conversions to be done
                                  192
    20DA                          193  Down_load:
    20DA B12800              E    194          ldb     control_a2d,#dump_b0_p_s        ; download b0 paired/signed
    20DD EF0000              E    195          CALL    a2d_buff_util
    20E0 F0                       196          RET
                                  197
                                  198  ;-------------------------------------------------------------------------
    20E1                          199  TABLE_LOAD:
    20E1 0120                     200          clr     bx
    20E3 A1022110                 201          ld      ax,#DATA0       ; Load tabled data for testing
    20E7 A21D22                   202  load:   ld      cx,[ax]+
    20EA A21D1E                   203          ld      dx,[ax]+
    20ED C321800022               204          st      cx,xreal[bx]
    20F2 C321C0001E               205          st      dx,ximag[bx]
    20F7 65020020                 206          add     bx,#2
    20FB 89400020                 207          cmp     bx,#64
    20FF DEE6                     208          blt     LOAD
    2101 F0                       209          RET
                                  210
    2102                          211  DATA0:          ; SQUARE WAVE
                                  212
    2102 FF7FFF7FFF7FFF7F         213  DCW      32767,  32767,  32767,  32767,  32767,  32767,  32767,  32767
    2112 FF7FFF7FFF7FFF7F         214  DCW      32767,  32767,  32767,  32767,  32767,  32767,  32767,  32767
    2122 FF7FFF7FFF7FFF7F         215  DCW      32767,  32767,  32767,  32767,  32767,  32767,  32767,  32767
    2132 FF7FFF7FFF7FFF7F         216  DCW      32767,  32767,  32767,  32767,  32767,  32767,  32767,  32767
    2142 0180018001800180         217  DCW     -32767, -32767, -32767, -32767, -32767, -32767, -32767, -32767
    2152 0180018001800180         218  DCW     -32767, -32767, -32767, -32767, -32767, -32767, -32767, -32767
    2162 0180018001800180         219  DCW     -32767, -32767, -32767, -32767, -32767, -32767, -32767, -32767
    2172 0180018001800180         220  DCW     -32767, -32767, -32767, -32767, -32767, -32767, -32767, -32767
                                  221
                                  222  $eject
```

270189-49

Listing 3—Main Routine (Continued)

```
ERR LOC  OBJECT              LINE      SOURCE STATEMENT
                              223  ;-------------------------------------------------------------------------
    2182                      224  DO_WINDOW:                          ;;;; PERFORM HANNING WINDOW
    2182 012C                 225          clr     wndptr
    2184 012E                 226          clr     varptr              ; Windowing provides an effective
    2186                      227  WINDOW:                             ; divide by 2 because of the multiply
    2186 A32DBE211C           228          ld      ax,hanning[wndptr]
    218B A32DC02120           229          ld      bx,hanning+2[wndptr]
    2190 FE4F2FB0001C24       230          mul     tmpreal,ax,xreal[varptr]
    2197 FE4F2FC0002028       231          mul     tmpimag,bx,ximag[varptr]
    219E 0D0124               232          shll    tmpreal,#1
    21A1 0D0128               233          shll    tmpimag,#1                  ; Compensate for the divide by 2
    21A4 C32FB00026           234          st      tmpreal+2,xreal[varptr]
    21A9 C32FC0002A           235          st      tmpimag+2,ximag[varptr]
    21AE 6504002C             236          add     wndptr,#4
    21B2 6502002E             237          add     varptr,#2
    21B6 8940002E             238          cmp     varptr,#64
    21BA D7CA                 239          jne     window
    21BC F0                   240          RET
                              241
    21BE                      242  HANNING:        ; Windowing function
                              243
    21BE 00004F003B01C102     244  DCW          0,   79,  315,  705, 1247, 1935, 2761, 3719
    21CE BF126617711CD421     245  DCW       4799, 5990, 7281, 8660,10114,11628,13187,14778
    21DE 004045467C4C9352     246  DCW      16384,17989,19580,21139,22653,24107,25486,26777
    21EE 406D787136757078     247  DCW      27968,29048,30006,30832,31520,32062,32452,32688
    21FE FF7FB07FC47E3E7D     248  DCW      32767,32688,32452,32062,31520,30832,30006,29048
    220E 406D99688E632B5E     249  DCW      27968,26777,25486,24107,22653,21139,19580,17989
    221E 0040BA3983336C2D     250  DCW      16384,14778,13187,11628,10114, 8660, 7281, 5990
    222E BF12870EC90A8F07     251  DCW       4799, 3719, 2761, 1935, 1247,  705,  315,   79
    223E 0000                 252  DCW          0
                              253
                              254  $eject
```

270189-50

Listing 3—Main Routine (Continued)

```
MCS-96 MACRO ASSEMBLER    FFT_MAIN_APNOTE                    02/18/86      PAGE    7

ERR LOC  OBJECT                  LINE        SOURCE STATEMENT
                                  255
    3D00                          256   CSEG AT 3D00H          ; ADDITIONAL  TABLES  FOR  TESTING
                                  257                 ; SINE 7.0 X
    3D00                          258   DATA1:
    3D00 00003351897DE270         259   DCW         0, 20787, 32137, 28898, 12539, -9512,-27245,-32609
    3D10 7EA574F31C477C7A         260   DCW     -23170, -3212, 18204, 31356, 30273, 15446, -6393,-25329
    3D20 01800F9D08E7563C         261   DCW     -32767,-25329, -6392, 15446, 30273, 31356, 18204, -3212
    3D30 7EA59F809395D8DA         262   DCW     -23170,-32609,-27245, -9512, 12539, 28898, 32137, 20787
    3D40 0000CDAE77821E8F         263   DCW        -0,-20787,-32137,-28898,-12539,  9512, 27245, 32609
    3D50 825A8C0CE4B88485         264   DCW      23170,  3212,-18204,-31356,-30273,-15446,  6393, 25329
    3D60 FF7FF162F818AAC3         265   DCW      32767, 25329,  6392,-15446,-30273,-31356,-18204,  3212
    3D70 825A617F6D6A2825         266   DCW      23170, 32609, 27245,  9512,-12539,-28898,-32137,-20787
                                  267
    3D80                          268   DATA2:          ; SINE 7.5 X
                                  269
    3D80 0000F555617FCF66         270   DCW         0, 22005, 32609, 26319,  6393,-16846,-31356,-29621
    3D90 05CF1F2BE270297C         271   DCW     -12539, 11039, 28898, 31785, 18204, -4808,-25329,-32728
    3DA0 7EA5B8F933519C7E         272   DCW     -23170, -1608, 20787, 32412, 27245,  7962,-15446,-30852
    3DB0 BF8946C92825C96D         273   DCW     -30273,-14010,  9512, 28105, 32137, 19519, -3212,-24279
    3DC0 018029A174F33F4C         274   DCW     -32767,-24279, -3212, 19519, 32137, 28105,  9512,-14010
    3DD0 BF897C87AAC31A1F         275   DCW     -30273,-30852,-15446,  7962, 27245, 32412, 20787, -1608
    3DE0 7EA528800F9D38ED         276   DCW     -23170,-32728,-25329, -4808, 18205, 31785, 28898, 11039
    3DF0 05CF4B8C848533BE         277   DCW     -12539,-29621,-31356,-16845,  6393, 26319, 32609, 22005
                                  278
    3E00                          279   DATA3:          ; .707*SINE 7.5X
                                  280
    3E00 0000C63C0F5AAF48         281   DCW         0, 15558, 23055, 18607,  4520,-11910,-22169,-20942
    3E10 5FDD7C1ECF4FC857         282   DCW      -8865,  7804, 20431, 22472, 12870, -3399,-17908,-23138
    3E20 03C08FFB69398459         283   DCW     -16381, -1137, 14697, 22916, 19262,  5629,-10921,-21812
    3E30 65AC4FD9451A9E4D         284   DCW     -21403, -9905,  6725, 19870, 22721, 13800, -2271,-17165
    3E40 82A5F3BC21F7E835         285   DCW     -23166,-17165, -2271, 13800, 22721, 19870,  6725, -9905
    3E50 65ACCCAA58D5FD15         286   DCW     -21403,-21812,-10920,  5629, 19262, 22916, 14696, -1137
    3E60 03C09EA50CBAB9F2         287   DCW     -16381,-23138,-17908, -3399, 12871, 22472, 20431,  7804
    3E70 5FDD32AE67A97AD1         288   DCW      -8865,-20942,-22169,-11910,  4520, 18607, 23055, 15557
                                  289
    3E80                          290   DATA4:          ; .707*SINE(11x) /16
                                  291
    3E80 0000FD04B40472FF         292   DCW         0,  1277,  1204,  -142, -1338, -1119,   282,  1386
    3E90 00045CFE74FA69FC         293   DCW      1024,  -420, -1420,  -919,   554,  1441,   804,  -683
    3EA0 58FA55FD2403A105         294   DCW     -1448,  -683,   804,  1441,   554,  -919, -1420,  -420
    3EB0 00046A051A01A1FB         295   DCW      1024,  1386,   282, -1119, -1338,  -142,  1204,  1277
    3EC0 000003FB4CFB8E00         296   DCW        -0, -1277, -1204,   142,  1338,  1119,  -282, -1386
    3ED0 00FCA4018C059703         297   DCW     -1024,   420,  1420,   919,  -554, -1441,  -804,   683
    3EE0 A805AB02DCFC5FFA         298   DCW      1448,   683,  -804, -1441,  -554,   919,  1420,   420
    3EF0 00FC96FAE6FE5F04         299   DCW     -1024, -1386,  -282,  1119,  1338,   142, -1204, -1277
                                  300
    3F00                          301   DATA5:          ; .707*(SINE 7.5X + 1/16 SINE 11X)
```

270189-51

```
MCS-96 MACRO ASSEMBLER    FFT_MAIN_APNOTE                          02/18/86        PAGE    8

ERR LOC  OBJECT                LINE        SOURCE STATEMENT
                                302
    3F00 0000C241C35E2148       303   DCW       0, 16834, 24259, 18465,  3182,-13029,-21886,-19557
    3F10 5EE1D81C434A3154       304   DCW      -7842,  7384, 19011, 21553, 13425, -1958,-17103,-23821
    3F20 5BBAE5F88D3C245F       305   DCW     -17829, -1819, 15501, 24356, 19816,  4710,-12341,-22232
    3F30 65B0B9DE5F1B3F49       306   DCW     -20379, -8519,  7007, 18751, 21383, 13658, -1067,-15888
    3F40 82A5F6B76DF27636       307   DCW     -23166,-18442, -3475, 13942, 24059, 20990,  6442,-11290
    3F50 65A870ACE4DA9419       308   DCW     -22427,-21392, -9500,  6548, 18708, 21475, 13892,  -454
    3F60 ABC548A8E8B618ED       309   DCW     -14933,-22456,-18712, -4840, 12317, 23391, 21851,  8225
    3F70 5FD9C8A84DA8D9D5       310   DCW      -9889,-22328,-22451,-10791,  5857, 18749, 21851, 14281
                                311
                                312
    3F80                        313   END

ASSEMBLY COMPLETED,   NO ERROR(S) FOUND.
```

270189-52

Listing 3—Main Routine (Continued)

```
SERIES-III MCS-96 RELOCATOR AND LINKER, V2.0
Copyright 1983 Intel Corporation

INPUT FILES: :F2:FTMAIN.OBJ, :F2:FFTRUN.OBJ, :F2:PLOTSP.OBJ, :F2:A2DCON.OBJ
OUTPUT FILE: :F2:FFTOUT
CONTROLS SPECIFIED IN INVOCATION COMMAND:
   IX


INPUT MODULES INCLUDED:
   :F2:FTMAIN.OBJ(FFT_MAIN_APNOTE)  02/18/86
   :F2:FFTRUN.OBJ(FFT_RUN)  02/18/86
   :F2:PLOTSP.OBJ(PLOT_SERIAL)  02/18/86
   :F2:A2DCON.OBJ(A2D_BUFFERING_UTILITY)  02/18/86


SEGMENT MAP FOR :F2:FFTOUT(FFT_MAIN_APNOTE):

             TYPE    BASE      LENGTH    ALIGNMENT    MODULE NAME
             ----    ----      ------    ---------    -----------

**RESERVED*          0000H     001AH
             REG     001AH     0001H     BYTE         PLOT_SERIAL
*** GAP ***          001BH     0001H
             REG     001CH     0008H     ABSOLUTE     FFT_MAIN_APNOTE
             OVRLY   0024H     0035H     ABSOLUTE     FFT_RUN
**OVERLAP** OVRLY    0024H     0010H     ABSOLUTE     PLOT_SERIAL
**OVERLAP** OVRLY    0024H     000CH     ABSOLUTE     FFT_MAIN_APNOTE
*** GAP ***          0059H     0001H
             OVRLY   005AH     0006H     WORD         A2D_BUFFERING_UTILITY
             REG     0060H     000CH     WORD         A2D_BUFFERING_UTILITY
             REG     006CH     0003H     BYTE         FFT_MAIN_APNOTE
*** GAP ***          006FH     0011H
             DATA    0080H     0080H     ABSOLUTE     FFT_MAIN_APNOTE
             STACK   0100H     001EH     WORD
             DATA    011EH     0080H     WORD         FFT_RUN
*** GAP ***          019EH     0062H
             DATA    0200H     0140H     ABSOLUTE     FFT_MAIN_APNOTE
*** GAP ***          0340H     1CC2H
             CODE    2002H     0002H     ABSOLUTE     A2D_BUFFERING_UTILITY
*** GAP ***          2004H     007CH
             CODE    2080H     01C0H     ABSOLUTE     FFT_MAIN_APNOTE
*** GAP ***          2240H     0040H
             CODE    2280H     0215H     ABSOLUTE     FFT_RUN
*** GAP ***          2495H     006BH
             CODE    2500H     0168H     ABSOLUTE     PLOT_SERIAL
             CODE    2668H     00ECH     BYTE         A2D_BUFFERING_UTILITY
*** GAP ***          2754H     10ACH
             CODE    3800H     042AH     ABSOLUTE     FFT_RUN
*** GAP ***          3C2AH     00D6H
             CODE    3D00H     0280H     ABSOLUTE     FFT_MAIN_APNOTE
*** GAP ***          3F80H     C080H
```

270189-53

Listing 3—Main Routine (Continued)

Several constants are then setup for other routines. The purpose of centrally locating these constants was the ease of modifying the operation of the routines. Note that AVR_NUM and SCALE_FACTOR must be changed at the same time. SCALE_FACTOR is the shift count used to divide each FFT output value before it is added to the output array. AVR_NUM must be less than 2**SCALE_FACTOR or an overflow could occur. Next, the public variables are declared for the arrays and a few other parameters.

The program then begins by setting the stack pointer and waiting for the SBE-96 to finish talking to the terminal. If this is not done, there may be serial port interrupts occurring for the first twenty five milliseconds of program operation.

Initialization of the plotter is next, followed by setting the FFT_MODE byte. This byte controls the graphing, loading and magnitude calculation of the FFT data. Since FFT_MODE is declared PUBLIC in this module, and EXTERNAL in the PLOT module and FFTRUN module, the extra bits available in this byte can be used for future enhancements.

The next step is to clear the FFT output array. Since the FFT program can be set to average its results by dividing the output before adding it to the magnitude array, the array must be cleared before beginning the program.

Data is then loaded into into the FFT input array by the code at LOAD_DATA, or the code at TABLE_LOAD, depending on the value of FFT_MODE bit 0. The tabled data located at DATA0 is a square wave of magnitude 1. This waveform provides a reasonable test of the FFT algorithm, as many harmonics are generated. The results are also easy to check as the pattern contains half zeros, imaginary values which are always the same, and real values which decrease. Figure 13 shows the output in fractions, hexadecimal and decimal. The hexadecimal and decimal values are based on an output of 16384 being equal to 1.00.

Note that the magnitude is

$$SQR\ (REAL^2 + IMAG^2)$$

and the dB value is

$$10\ LOG\ (\ (REAL^2 + IMAG^2)/65536\ )$$

The divide by 65536 is used for the dB scale to provide a reasonable range for calculations. If this was not done, a 32-bit LOG function would have been needed.

After the data is loaded, the data is optionally windowed, based on FFT_MODE bit 1, and the FFT program is called. Once the loop has been performed AVR_CNT times, the graph is drawn by the plot routine.

Appended to the main routine is the FFTOUT.M96 Listing. This is provided by the relocator and linker, RL96. With this listing and the main program, it is possible to determine which sections of code are at which addresses.

Using the modular programming methods employed here, it is reasonably easy to debug code. By emulating the program in a relatively high level language, each routine can be checked for functionality against a known standard. The closer the high level implementation matches the ASM96 version, the more possible checkpoints there are between the two routines.

Once all of the program routines (modules) can be shown to work individually, the main program should work unless there is unwanted interaction between the modules. These interactions can be checked by verifying the inputs and outputs of each module. The assembly language locations to perform the program breaks can be retrieved by absolutely locating the main module. The other modules can be dynamically located by RL96.

The more interactive program modules are, the more difficult the program becomes to debug. This is especially true when multiple interrupts are occurring, and several of the interrupt routines are themselves interruptable. In these cases, it may be necessary to use debugging equipment with trace capability, like the VLSiCE-96. If this type of equipment is not available, then using I/O ports to indicate the entering and leaving of each routine may be useful. In this way it will be possible to watch the action of the program on an oscilloscope or logic analyzer. There are several places within this code that I/O port toggling has been used as an aid to debugging the program. These lines of code are marked "FOR INDICATION ONLY."

K	Fractional			dB	Decimal			Hexadecimal		
	REAL	IMAG	MAG²		REAL	IMAG	MAG²	REAL	IMAG	MAG²
0	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
1	0.0625	−1.2722	1.2738	38.225	1024	−20843	20868	400	AE95	5184
2	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
3	0.0625	−0.4213	0.4260	28.710	1024	−6903	6978	400	E509	1B42
4	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
5	0.0625	−0.2495	0.2572	24.329	1024	−4088	4214	400	F008	1076
6	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
7	0.0625	−0.1747	0.1855	21.491	1024	−2862	3039	400	F4D2	BDF
8	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
9	0.0625	−0.1321	0.1462	19.421	1024	−2165	2395	400	F78B	95B
10	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
11	0.0625	−0.1043	0.1216	17.820	1024	−1708	1992	400	F954	7C8
12	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
13	0.0625	−0.0843	0.1049	16.540	1024	−1381	1719	400	FA9B	6B7
14	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
15	0.0625	−0.0690	0.0931	15.499	1024	−1130	1525	400	FB96	5F5
16	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
17	0.0625	−0.0566	0.0844	14.645	1024	−928	1382	400	FC60	566
18	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
19	0.0625	−0.0464	0.0778	13.944	1024	−759	1275	400	FD09	4FB
20	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
21	0.0625	−0.0375	0.0729	13.374	1024	−614	1194	400	FD9A	4AA
22	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
23	0.0625	−0.0296	0.0691	12.918	1024	−484	1133	400	FE1C	46D
24	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
25	0.0625	−0.0224	0.0664	12.564	1024	−366	1088	400	FE92	440
26	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
27	0.0625	−0.0157	0.0644	12.305	1024	−256	1056	400	FF00	420
28	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
29	0.0625	−0.0093	0.0632	12.135	1024	−152	1035	400	FF68	40B
30	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
31	0.0625	−0.0031	0.0626	12.051	1024	−50	1025	400	FFCE	401

Figure 13. FFT Output for a Square Wave Input

9.0 ANALOG TO DIGITAL CONVERTER MODULE

The module presented in Listing 4 is a general purpose one which converts analog values under interrupt control and stores them in one of two buffers. These buffers can then be downloaded to another buffer, such as the input buffer to the FFT program. During downloading, this module can convert the data into signed or unsigned formats, and fill a linear or a paired array. A paired array is like the one used in the FFT transform program. It requires N data points placed alternately in two arrays, one starting at zero and the other at N/2.

```
MCS-96 MACRO ASSEMBLER    A2D_BUFFERING_UTILITY                    02/18/86        PAGE    1

SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE: :F2:A2DCON.A96
OBJECT FILE: :F2:A2DCON.OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB

ERR LOC  OBJECT                 LINE        SOURCE STATEMENT
                                   1    $pagelength(50)
                                   2
                                   3    A2D_Buffering_Utility   module  stacksize(12)
                                   4    ;
                                   5    ; Intel Corporation,  July 16, 1985
                                   6    ; by Dave Ryan, Intel Applications Engineer
                                   7    ;
                                   8    ; This utility fills a memory buffer with A/D conversion results.  The
                                   9    ; conversions are done under interrupt control, and are initiated when
                                  10    ; A2D_BUFF_Util is called.  The results of the conversions are placed
                                  11    ; in one of two buffers, called BUFF0 and BUFF1.
                                  12    ;
                                  13    ; This utility provides options for the selection of the buffer lengths, data
                                  14    ; format, sample period, conversion channel and time base.  The utility also
                                  15    ; has a donwload routine that will load either buffer into a register file
                                  16    ; buffer.  Output formats can also be chosen for the downloaded buffer.  The
                                  17    ; data can be formatted as signed or unsigned linear or paried arrays.
                                  18    ;
                                  19    ; RUN-TIME OPTIONS
                                  20    ;
                                  21    ; Rather than use the STACK to pass controls, this utility gets its directions
                                  22    ; from 2 control words in memory.  The utility expects that its control words
                                  23    ; are valid at the time A2D_BUFF_Util is called and remain valid throughout
                                  24    ; A/D interrupt executions and downloads.  The control words are:
                                  25    ;
                                  26    ;       Sample_Period   ; WORD ; The time between samples in timer counts
                                  27    ;                              ; where the timer used has been specified
                                  28    ;
                                  29    ;
                                  30    ;       Control_A2D     ; BYTE ; Control information for the utility:
                                  31    ;                                BIT#
                                  32    ;
                                  33    ;                              ;  0-2  ; Channel Number
                                  34    ;;                             ;   3   ; Signed Result/Unsigned Result#
                                  35    ;                              ;   4   ; Convert/Download#
                                  36    ;                              ;   5   ; BUFF1/BUFF0# for conversions
                                  37    ;                              ;       ; BUFF0/BUFF1# for downloads
                                  38    ;                              ;   6   ; Linear/Paired#
                                  39    ;                              ;   7   ; Converter BUSY/IDLE#
                                  40
                                  41    $EJECT
```

270189-54

```
MCS-96 MACRO ASSEMBLER    A2D_BUFFERING_UTILITY                    02/18/86        PAGE    2

ERR LOC  OBJECT                LINE        SOURCE STATEMENT
                                 42    ;
                                 43    ; The following is a table of equates that can be used to simplify the
                                 44    ; bit diddling requirements.  If you are not running conversions concurrently
                                 45    ; with downloads, always LDB Control_A2D with the following command then
                                 46    ; ORB Control_A2D with the channel number you wish to convert if you are
                                 47    ; starting a conversion.
                                 48    ;
                                 49    ; Once the utility is called, care must be taken when Control_A2d is
                                 50    ; modified.  You can cause downloads to occur while conversions are running,
                                 51    ; but you cannot start conversions during a download.  To do this, ORB to the
                                 52    ; control byte with the appropriate bits set.  Do NOT change the BUFF bit or
                                 53    ; the BUSY bit.  Just set the download bit and set the data format bits to the
                                 54    ; correct values.
                                 55    ;
                                 56    ; The BUFF bit has opposite definitions for conversions and downloads.  This
                                 57    ; allows conversions to be done into BUFF0 while downloads come from BUFF1, and
                                 58    ; vice versa.
                                 59    ;
                                 60    ; A2D UTILITY COMMANDS_____________________________________________________________
                                 61    ;
                                 62    ;con_b0          equ     00010000b;convert to BUFF0
                                 63    ;con_b1          equ     00110000b;    "       BUFF1
                                 64    ;
                                 65    ;dump_b0_l_u     equ     01100000b; download  BUFF0  as LINEAR USIGNED data
                                 66    ;dump_b1_l_u     equ     01000000b;    "      BUFF1  "    "      "       "
                                 67    ;dump_b0_p_u     equ     00100000b;    "      BUFF0  "  PAIRED   "       "
                                 68    ;dump_b1_p_u     equ     00000000b;    "      BUFF1  "    "      "       "
                                 69    ;dump_b0_l_s     equ     01101000b; download  BUFF0  as LINEAR SIGNED  data
                                 70    ;dump_b1_l_s     equ     01001000b;    "      BUFF1  "    "      "       "
                                 71    ;dump_b0_p_s     equ     00101000b;    "      BUFF0  "  PAIRED   "       "
                                 72    ;dump_b1_p_s     equ     00001000b;    "      BUFF1  "    "      "       "
                                 73    ;___________________________________________________________________________
                                 74    $eject
```

270189-55

```
MCS-96 MACRO ASSEMBLER    A2D_BUFFERING_UTILITY                    02/18/86        PAGE    3

ERR LOC  OBJECT                LINE        SOURCE STATEMENT
                                 75  ;
                                 76  ; ASSEMBLY-TIME OPTIONS
                                 77  ;
                                 78  ; The base addresses and length of each conversion buffer and the destination
                                 79  ; buffer are DECLARED EXTRNal in this utility. Other options such as selection
                                 80  ; of the timer used as a timebase, the length of the buffer, and the effective
                                 81  ; number of bits in the reported result are set at assembly time through use
                                 82  ; of EQUates in this module.
                                 83  ;
                                 84  ; The following parameters need to be provided at assembly or link time.
                                 85  ; The buffer bases are declared EXTRNal by this utility, while the buffer
                                 86  ; length shift count and HSO commands are EQUated.
                                 87  ;
                                 88  ;       BUFF0_BASE      ; The starting address of BUFF0
                                 89  ;       BUFF1_BASE      ; The starting address of BUFF1
                                 90  ;       DEST_BUFF_BASE  ; The starting address of the download
                                 91  ;                               ; target buffer.
                                 92  ;
                                 93  ;       BUFF_LENGTH     ; The number of SAMPLES that each
                                 94  ;                               ; buffer must hold. must be >1 and <256
                                 95  ;
                                 96  ;       Shift_count     ; The number of times that the conversion result is
                                 97  ;                       ; to be shifted right from its natural left justified
                                 98  ;                       ; position. Setting a shift count greater than 6 will
                                 99  ;                       ; result in lost bits to the right.  Rounding is NOT
                                100  ;                       ; done.
                                101  ;
                                102  ;       CLOCK           ; Specify as either TIMER1 or T2CLK.  This is the
                                103  ;                       ; timebase used for conversions.
                                104  ;
                                105  ;       Samples are stored as words in the buffers.  The program stores
                                106  ; conversions linearly in BUFF0 and BUFF1, and linearly or paired in the
                                107  ; destination buffer as selected.  If the download is to be paired, the first
                                108  ; sample is placed in location DEST_BUFF_BASE, the second sample is placed in
                                109  ; location (DEST_BUFF_BASE + BUFF_LENGTH), the third in (DEST_BUFF_BASE + 2),
                                110  ; the fourth in (DEST_BUFF_BASE + 2 + BUFF_LENGTH), etc.
                                111  ;
                                112  $eject
```

270189-56

Listing 4—A to D Converter Routine (Continued)

```
ERR LOC  OBJECT          LINE      SOURCE STATEMENT
                          113  ;
                          114  ;NOTES ON EXECUTION
                          115  ;
                          116  ; When a utility call directs the initiation of a set of A2D conversions, the
                          117  ; first conversion is begun at approximately one sample time plus 50 state
                          118  ; times from when the utility was called.  This assumes that no interrupts are
                          119  ; present.
                          120  ;
                          121  ; The conversion busy bit is set approximately 50 state times after a call
                          122  ; to the utility, if the convert bit was set in the A2D_Control byte.  The
                          123  ; busy bit is cleared after all conversion results have been stored in the
                          124  ; result buffer designated (BUFF0 or BUFF1).
                          125  ;
                          126  ; Take great care in modifying the A2D_Control byte to do a download while
                          127  ; conversions are taking place.  You can never download a buffer that is
                          128  ; being converted into.  The results would be invalid.
                          129  $eject
```

```
MCS-96 MACRO ASSEMBLER    A2D_BUFFERING_UTILITY                    02/18/86        PAGE    5

ERR LOC  OBJECT              LINE        SOURCE STATEMENT
                             130
    0000                     131           RSEG
                             132
                             133     EXTRN BUFF0_BASE, BUFF1_BASE, DEST_BUFF_BASE
                             134     EXTRN ad_command, ad_result_lo, ad_result_hi
                             135     EXTRN hso_command, hso_time,sp
                             136
      0040                   137     BUFF_LENGTH     EQU     64
      0001                   138     Shift_Count     EQU     1
      000A                   139     CLOCK           EQU     TIMER1
                             140
                             141     ; set up hso commands for correct timer *********************************
                             142
      000A                   143     TIMER1          equ     0AH
      000C                   144     T2CLK           equ     0CH
                             145
      0000                   146     MASK            equ     (10h*CLOCK)AND(40h)
                             147
      000F                   148     Start_A2D       equ     (00001111b)OR(MASK)
                             149                             ;start a2d based on timer 1, no interrupt
                             150
      0000                   151     HSO_0_Low       equ     (00000000b)OR(MASK)
                             152                             ; make hso.0 low based on timer1 no interrupt
                             153
      0020                   154     HSO_0_High      equ     (00100000b)OR(MASK)
                             155                             ; make hso.0 hi  based on timer1 no interrupt
                             156
                             157
                             158     ; set up storage ********************************************************
                             159
    0000                     160     adudtemp0:      DSW     1; temp register for download calls
                             161
    0002                     162     aductemp0:      DSW     1; temp registers for conversion calls
    0004                     163     aductemp1:      DSW     1
    0006                     164     top_of_buffer:  DSW     1
    0008                     165     sample_count:   DSB     1
                             166
    0009                     167     Control_A2D:    DSB     1; the byte that controls the utility execution
      0003                   168                     DForm   equ     3       ; Signed/Unsigned#
      0004                   169                     Con_Dwn equ     4       ; Convert/Download#
      0005                   170                     B0_B1   equ     5       ; Buff1/Buff0# for conversions
                             171                                             ; Buff0/Buff1# for downloads
      0006                   172                     Lin_Par equ     6       ; Linear/Paired#
      0080                   173                     Busy    equ     10000000B       ; Bit 8
                             174     $eject
```

270189-58

```
MCS-96 MACRO ASSEMBLER    A2D_BUFFERING_UTILITY                    02/18/86        PAGE    6

ERR LOC  OBJECT              LINE       SOURCE STATEMENT
                             175
    000A                     176  Sample_Period:  DSW     1; the word that specifies the number of clock ticks
                             177                           ; that elapse between each sample
                             178
                             179  PUBLIC  Control_A2D, Sample_Period
                             180
                             181
    0000                     182          OSEG
                             183
    0000                     184  src_ptr:        DSW     1; some overlayable temp registers
      0000                   185          temp set src_ptr:WORD
    0002                     186  dest_ptr:       DSW     1
    0004                     187  loop_count:     DSW     1
                             188
                             189
    2002                     190          CSEG    at      2002h
                             191
                             192  PUBLIC  A2D_DONE_Vector
                             193
    2002 AC00          R     194  DCW     A2D_DONE_Vector
                             195
                             196
    0000                     197          CSEG
                             198
                             199  PUBLIC  A2D_BUFF_Util
                             200
                             201  Load_HSO_Command MACRO  var     ; Macro to load HSO
                             202
                             203          LDB     hso_command,#var
                             204          LD      hso_time,aductemp0
                             205  ENDM
                             206  $eject
```

270189-59

Listing 4—A to D Converter Routine (Continued)

```
MCS-96 MACRO ASSEMBLER    A2D_BUFFERING_UTILITY                    02/18/86          PAGE    7

ERR LOC  OBJECT              LINE       SOURCE STATEMENT
                              207
    0000                      208   A2D_BUFF_Util:
                              209
    0000 3C0962          R    210           JBS     Control_A2D, Con_Dwn, Convert   ; Select convert or download
    0003                      211   Download:
    0003 A1000000        E    212           LD      src_ptr,#BUFF1_BASE
    0007 350904          R    213           JBC     Control_A2D, BO_Bl,   Set_Data_Format
                              214
    000A                      215   Download_BUFF0:
    000A A1000000       E     216           LD      src_ptr,#BUFF0_BASE
                              217
                              218
    000E                      219   Set_Data_Format:                        ; Choose linear or paired
    000E A1000002       E     220           LD      dest_ptr, #DEST_BUFF_BASE
    0012 B14004         R     221           LDB     loop_count,#BUFF_LENGTH
    0015 3E091D         R     222           JBS     Control_A2D, Lin_Par,    Linear_data_loop
                              223
                              224
    0018 180104         R     225   PAIRED: SHRB    loop_count,#1           ; The paired data routine uses 1/2
                              226                                           ; as many loops as the unpaired
    001B                      227   Paired_Data_loop:
    001B A20000         R     228           LD      adudtemp0,[src_ptr]+            ; Move even word
    001E C20200         R     229           ST      adudtemp0,[dest_ptr]
    0021 65400002       R     230           ADD     dest_ptr,#BUFF_LENGTH   ; Length = # of words = 1/2 # of bytes
                              231
    0025 A20000         R     232           LD      adudtemp0,[src_ptr]+            ; Move odd word
    0028 C20200         R     233           ST      adudtemp0,[dest_ptr]+
    002B 69400002       R     234           SUB     dest_ptr,#BUFF_LENGTH
                              235
    002F E004E9         R     236           DJNZ    loop_count, Paired_Data_loop    ; Loop until done
                              237
    0032 280D                 238           CALL    Convert_Data
    0034 F0                   239           RET
                              240
                              241
    0035                      242   Linear_Data_loop:                       ; Move data linearly
    0035 A20000         R     243           LD      adudtemp0,[src_ptr]+
    0038 C20200         R     244           ST      adudtemp0,[dest_ptr]+
                              245
    003B E004F7         R     246           DJNZ    loop_count, Linear_Data_loop    ; Loop until done
                              247
    003E 2801                 248           CALL    Convert_Data
    0040 F0                   249           RET
                              250   $eject
```

270189-60

Listing 4—A to D Converter Routine (Continued)

```
MCS-96 MACRO ASSEMBLER    A2D_BUFFERING_UTILITY                    02/18/86        PAGE    8

ERR LOC  OBJECT                LINE       SOURCE STATEMENT
                               251
    0041                       252   Convert_Data:                ; Convert the data in the destination buffer
                               253
    0041 A1400004          R   254           LD      loop_count,#BUFF_LENGTH
    0045 A1000000          E   255           LD      src_ptr,#DEST_BUFF_BASE
                               256
    0049 A20000            R   257   Again:  LD      adudtemp0,[src_ptr]
    004C 71C000            R   258           ANDB    adudtemp0,#11000000b
    004F 330909            R   259           JBC     Control_A2D, DForm, Unsigned_Result
                               260
    0052                       261   Signed_Result:
    0052 69E07F00          R   262           SUB     adudtemp0,#7fe0H
    0056 0A0100            R   263           SHRA    adudtemp0,#Shift_Count
    0059 2003                  264           BR      Replace_Sample
                               265
    005B                       266   Unsigned_Result:
    005B 080100            R   267           SHR     adudtemp0, #Shift_Count
                               268
    005E                       269   Replace_Sample:
    005E C20000            R   270           ST      adudtemp0,[src_ptr]+
    0061 E004E5            R   271           DJNZ    loop_count,Again        ; Loop until done
                               272
    0064 F0                    273           RET
                               274
                               275
    0065                       276   Convert:        ;; Prepare to Start Conversions
                               277
    0065 F2                    278           PUSHF
                               279
    0066 918009            R   280           ORB     Control_A2D, #Busy              ; set converter busy bit
                               281
    0069 B13F08            R   282           LDB     sample_count,#BUFF_LENGTH - 1
    006C A1000006          E   283           LD      top_of_buffer,#BUFF0_BASE
    0070 A1800004          E   284           LD      aductempl,#(BUFF0_BASE + 2*BUFF_LENGTH)
                               285
    0074 350908            R   286           JBC     Control_A2D, B0_B1, Start_Conversions
    0077 A1000006          E   287           LD      top_of_buffer,#BUFF1_BASE
    007B A1800004          E   288           LD      aductempl,#(BUFF1_BASE + 2*BUFF_LENGTH)
                               289   $eject
```

270189-61

Listing 4—A to D Converter Routine (Continued)

```
MCS-96 MACRO ASSEMBLER    A2D_BUFFERING_UTILITY                    02/18/86        PAGE    9

ERR LOC  OBJECT              LINE       SOURCE STATEMENT
                              290
    007F                      291   Start_Conversions:
                              292
    007F 51070900         E   293         ANDB    ad_command,Control_A2D,#00000111b     ;load channel number
                              294
    0083 440A0A02         R   295         ADD     aductemp0,CLOCK,Sample_Period         ;start first conversion
                              296                                                       ;one sample time from
                              297                                                       ;now
                              298
                              299         Load_HSO_Command Start_A2D            ; Start A2D at Time=aductemp0
                              303
    008D CC00             R   304         POP     temp                          ; get a copy of the psw
                              305
                              306         Load_HSO_Command HSO_0_high           ; set hso.0 high at conversion
                              310                                               ; start time for external S/H
                              311
    0095 81020200         R   312         OR      temp,#202h                    ; enable a2d interrupts
                              313
    0099 640A02           R   314         ADD     aductemp0,Sample_Period
                              315
                              316         Load_HSO_Command Start_A2D            ; start second convertion one
                              320                                               ; sample time from the first
                              321
    00A2 C800             R   322         PUSH    temp                          ; put psw back on stack
                              323
                              324         Load_HSO_Command HSO_0_low            ;lower hso.0 for external S/H
                              328
    00AA F3                   329         POPF
    00AB F0                   330         RET
                              331   $eject
```

270189-62

```
MCS-96 MACRO ASSEMBLER    A2D_BUFFERING_UTILITY                    02/18/86        PAGE   10

ERR LOC  OBJECT              LINE       SOURCE STATEMENT
    00AC                      332           CSEG
                              333
    00AC                      334   A2D_DONE_Vector:              ; A/D INTERRUPT ROUTINE
    00AC F2                   335           PUSHF
                              336
    00AD C60600          E    337           STB     ad_result_lo,[top_of_buffer]+
    00B0 C60600          E    338           STB     ad_result_hi,[top_of_buffer]+
    00B3 51070900        E    339           ANDB    ad_command,Control_A2D,#00000111b       ;load channel number
                              340
    00B7 E00809          R    341           DJNZ    sample_count, Sample_Again
    00BA 1708            R    342           INCB    sample_count
                              343
    00BC 880406          R    344           CMP     top_of_buffer,aductempl         ; Check top of buffer
    00BF DF26                 345           BE      Top_of_buffers
    00C1 F3                   346           POPF
    00C2 F0                   347           RET
                              348
    00C3                      349   Sample_Again:
    00C3 640A02          R    350           ADD     aductemp0,Sample_Period         ; Set next sample time
    00C6 880406          R    351           CMP     top_of_buffer,aductempl         ; Check top of buffer
                              352                                                   ; for later jump
                              353           Load_HSO_Command Start_A2D
                              357
    00CF 30080B          R    358           JBC     sample_count,0,Make_HSO_High
                              359
    00D2                      360   Make_HSO_low:
    00D2 FD                   361           nop                             ; wait 8 states after HSO load
                              362           Load_HSO_Command HSO_0_Low
                              366                                           ; Load for change of HSO to trigger S/H
    00D9 DF0C                 367           BE      Top_of_buffers
    00DB F3                   368           POPF
    00DC F0                   369           RET
                              370
    00DD                      371   Make_HSO_high:
                              372           Load_HSO_Command HSO_0_High     ; Load for change of HSO to trigger S/H
                              376
    00E3 DF02                 377           BE      Top_of_buffers
    00E5 F3                   378           POPF
    00E6 F0                   379           RET
                              380
    00E7                      381   Top_of_buffers:
    00E7 717F09          R    382           ANDB    Control_A2D,#NOT(Busy)  ; Clear converter BUSY bit
    00EA F3                   383           POPF
    00EB F0                   384           RET
    00EC                      385   END

ASSEMBLY COMPLETED,   NO ERROR(S) FOUND.
```

270189-63

Listing 4—A to D Converter Routine (Continued)

The listing contains a fairly complete description of what the program does. The block by block operations are shown below:

Lines 1-198 describe the program, declare the variables and set up equates. Several of these variables are declared as overlayable, so the user needs to be careful if using this module for other than the FFT program.

Lines 205-210 declare a macro which is used to load the HSO unit. This will be used repeatedly through the code.

Lines 212-253 determine whether a conversion or download has been requested. If a download has been requested, the data is downloaded to the destination array as either paired or linear data. Paired data has been described earlier.

Lines 255-278 contain a subroutine which converts the destination array to either signed or unsigned numbers. The numbers are also shifted right to provide the desired full-scale value as requested by SHIFT_COUNT.

Lines 279-334 initialize the conversion routine. HSO.0 is toggled with the start of each routine so that an external sample and hold can be used. The instructions in lines 308, 316, and 326 have been interweaved with the Load_HSO_Commands to provide the required 8 state delays between HSO loadings. If this was not done, NOPs would have been needed. It is easier to understand the code if these lines are thought of as being gathered at line 326.

Lines 337-353 are the actual A/D interrupt routine. The A/D results are placed BYTE by BYTE on the buffer, the A/D is reloaded, and then the number of samples taken is compared to the number needed. Note that the A/D command register needs to be reloaded even if the channel does not change. INCB on line 348 is used to insure that the DJNZ falls through on the next pass (if sample_count is not reset).

Lines 355-396 complete the routine. The HSO is set up to trigger the next conversion and provide the HSO.0 toggle for an external sample and hold. Once again, the time between consecutive loads of the HSO is 8 states minimum. Note that this section of code has been optimized for speed by reducing branches to an absolute minimum and duplicating code where needed.

This concludes the description of the A to D buffer module. In the FFT program, this module is run, then the FFT transform module, then the plot module. This allows variables to be overlaid, saving RAM space. The time cost for this is not bad, considering the printer is the limiting factor in these conversions. If more RAM was provided, and the FFT was run with its data in external RAM, this module could be run simultaneously with the other modules.

10.0 DATA PLOTTING MODULE

The plot module is relatively straight-forward, and is shown in Listing 5. After the declarations, which include overlayable registers, an initialization routine is listed. This separately called routine sets up the serial port on the 8096 to talk to the printer. In this case, the port has to be set for 300 baud.

A console out routine follows. This routine can also be called by any program, but it is used only by the plot routine in this example. The write to port 1 is used to trace the program flow. The character to be output is passed to this routine on the stack. This conforms to PLM-96 requirements.

Since all stack operations on the 8096 are 16-bits wide, a multiple character feature has been added to the console out routine. If the high byte it receives is non-zero, the ASCII character in that byte is printed after the character in the low byte. If the high byte has a value between 128 and 255, the character in the low byte is repeated the number of times indicated by the least significant 7 bits of the high byte.

The print decimal number routine is next. It is called with two words on the stack. The first word is the unsigned value to be printed. The second byte contains information on the number of places to be printed and zero and blank suppression. This routine is not overflow-proof. The user must declare a sufficient number of places to be printed for all possible numbers.

The DRAW_GRAPH routine provides the plot. It first sends a series of carriage return, line feeds (CRLFs) to clear the printer and provides a margin on the paper. Each row is started with the row number, 2 spaces, and a "+". Asterisks are then plotted until

Number of asterisks > FFT Value / PLOT_RES

Recall that PLOT_RES is a variable set by the main program. When the number of asterisks hits the desired value, the value of the line is printed. If the Decibel mode is selected, the line value is divided by 512 and printed in integer + decimal part form, followed by "dB". If the number of asterisks reaches PLOT_MAX, no value is printed. The next line is then started. A line with only a "!" is printed before the next plot line to provide a more aesthetic display on the printer. If a CRT was used, this extra line would probably not be wanted.

```
MCS-96 MACRO ASSEMBLER    PLOT_SERIAL                          02/18/86        PAGE    1

SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE: :F2:PLOTSP.A96
OBJECT FILE: :F2:PLOTSP.OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB

ERR LOC  OBJECT                 LINE        SOURCE STATEMENT
                                  1   $pagelength(50)
                                  2
                                  3   PLOT_SERIAL  MODULE STACKSIZE (6)
                                  4
                                  5   ; Intel Corporation,  December 12, 1985
                                  6   ; by Ira Horden, MCO Applications
                                  7
                                  8   ;     This program produces a plot on serially connected printer.  The
                                  9   ; maginitude of each of the 32 input values is plotted horizontally, with one
                                 10   ; "!" followed by a linefeed between each plot line.  Each plot line starts
                                 11   ; with a "+" and the entire plot begins with 3 line feeds and ends with a form
                                 12   ; feed.  The values to be plotted are 32 unsigned words based at the externally
                                 13   ; defined pointer PLOT_IN.
                                 14
                                 15   ;     The routine INIT_OUTPUT must be run to set up the serial port when the
                                 16   ; system is turned on.  CON_OUT can be used by a program to output to the
                                 17   ; serial port.  DRAW_GRAPH is the routine that automatically plots the data.
                                 18
                                 19   ;     Sizing of the graph can be done using PLOT_RES, which determines how many
                                 20   ; units are needed for each dot, and PLOT_MAX, which is the maximum value the
                                 21   ; program will be passed.  Note that (PLOT_MAX/PLOT_RES) defines the maximum
                                 22   ; number of columns the routine will print.
                                 23   ;
                                 24
0000                             25   RSEG
                                 26           EXTRN   iocl, baud_reg, spcon, spstat, sbuf, portl
                                 27           EXTRN   zero, ax, bx, cx, dx, FFT_MODE
0000                             28           sptmp:  dsb     1
                                 29
0024                             30   OSEG at 24H
0024                             31           value:          dsl     1
0028                             32           divisor:        dsl     1
002C                             33           xptr:           dsw     1
002E                             34           yptr:           dsw     1
0030                             35           xval:           dsw     1
0032                             36           log_val:        dsw     1
                                 37
0000                             38   DSEG
                                 39           EXTRN   PLOT_IN
                                 40
                                 41   $eject
```

270189-64

```
ERR LOC  OBJECT              LINE      SOURCE STATEMENT
                               42
    2500                       43  CSEG at 2500H           ;;;;          PROGRAM  MODULE BEGINS
                               44
                               45   PUBLIC INIT_OUTPUT, CON_OUT, DRAW_GRAPH
                               46   EXTRN  PLOT_RES, PLOT_RES_2, PLOT_MAX
                               47
    2500                       48  INIT_OUTPUT:                        ; INITIALIZE SERIAL PORT
                               49
    2500 B12000          E     50          ldb     iocl,#00100000B ; set p2.0 to txd
                               51
      0270                     52  baud_val        equ     624             ; 624=300 baud (at 12 MHz)
                               53
        0082                   54  Baud_high       equ     ((baud_val-1)/256) OR 80H       ; set for XTAL1 clock
        006F                   55  baud_low        equ     (baud_val-1) MOD 256
                               56
    2503 B16F00          E     57          ldb     baud_reg,#baud_low
    2506 B18200          E     58          ldb     baud_reg,#baud_high
                               59
    2509 B14900          E     60          ldb     spcon,#01001001b        ; enable reciver mode 1
    250C B12000          R     61          ldb     sptmp,#00100000B        ; set TI-tmp
                               62
    250F F0                    63          RET
                               64
                               65  $eject
```

```
MCS-96 MACRO ASSEMBLER    PLOT_SERIAL                          02/18/86        PAGE    3

ERR LOC  OBJECT                LINE       SOURCE STATEMENT
                                 66
                                 67  ;-------------------------------------------------------------------------
                                 68  ;                          CONSOLE OUT ROUTINE
                                 69  ;
                                 70  ;       Call with a word parameter on stack.  The low byte has the character
                                 71  ;       to be sent.  If the high byte has a value between 81H and 8FEH, the
                                 72  ;       character is repeated 1 to 126 times respectively.  One repeat means
                                 73  ;       that the character will be printed 2 times.  If the high byte contains
                                 74  ;       a value between 1 and 7FH, the charater represented by that value will
                                 75  ;       be printed after the character in the low byte.  If the high byte
                                 76  ;       contains a value of zero only the low byte will be printed.
                                 77
     2510                        78  CON_OUT:
     2510 CC00                E  79          pop     ax                      ; cx contains the calling adress
     2512 CC00                E  80          pop     dx
     2514 3F011C              E  81          jbs     dx+1,7,onechr           ; If bit 7 is set print one character
     2517 980001              E  82          cmpb    dx+1,zero
     251A DF17                   83          je      onechr                  ; if highbyte=0 print one character
                                 84
     251C 900000              E  85  twochr: orb     sptmp,spstat            ; wait for TI
     251F 3500FA              R  86          jbc     sptmp,5,twochr
     2522 71DF00              R  87          andb    sptmp,#11011111b        ; clear TI-tmp
     2525 900000              E  88          orb     zero,spstat             ; remove possible false TI
                                 89
     2528 B00000              E  90          ldb     sbuf,dx
     252B B00100              E  91          ldb     dx,dx+1         ; Load second character
     252E 1101                E  92          clrb    dx+1            ; clear count byte
     2530 717F00              E  93          andb    dx,#07FH        ; mask MSB
                                 94
     2533 1701                E  95  onechr: incb    dx+1
     2535 717F01              E  96          andb    dx+1,#7FH
     2538 900000              E  97  wait1:  orb     sptmp,spstat            ; wait for TI
     253B 3500FA              R  98          jbc     sptmp,5,wait1
     253E 71DF00              R  99          andb    sptmp,#11011111b        ; clear TI-tmp
     2541 900000              E 100          orb     zero,spstat             ; remove possible false TI
                                101
     2544 B00000              E 102          ldb     sbuf,dx
     2547 E001EE              E 103          DJNZ    dx+1,wait1
     254A E300                E 104          BR      [ax]                    ; Effectively a RET
                                105
                                106  $eject
```

270189-66

```
ERR LOC  OBJECT                LINE      SOURCE STATEMENT
                               107   ;-------------------------------------------------------------------------
                               108   ;                          PRINT  DECIMAL  NUMBER  ROUTINE
                               109
                               110   ;       Call with two words on stack.  The first is the value to be printed.
                               111   ;       The second has mode information in the low byte.
                               112   ;               MODE:   000 = supress all zeros
                               113   ;                       001 = print all numbers
                               114   ;                       010 = supress all zeros except rightmost
                               115   ;                       1xx = do not print leading blanks
                               116   ;
                               117   ;       The high byte of the 2nd word = 2x the number of places to be printed
                               118
                               119
  254C                         120   PRINT_NUM:                      ; Send Decimal number to CON_OUT
  254C CC00              E     121           pop     cx
  254E CC00              E     122           pop     bx                ; bx is mode byte, bx+1 is divisor pointer
  2550 AC0100            E     123           ldbze   dx,bx+1
  2553 A300962528        E     124           ld      divisor,divtab[dx]
  2558 CC24                    125           pop     value
  255A                         126   div_loop:
  255A 0126                    127           clr     value+2
  255C 8C2824                  128           divu    value,divisor             ; divide ax,dx by divisor
  255F 380017            E     129           jbs     bx,0,chr_ok               ; print character regardless of value
  2562 980024            E     130           cmpb    value,zero
  2565 D70F                    131           jne     non_0                     ; jump if value is non zero
  2567                         132   Val_0:                          ; Value is zero
  2567 310003            E     133           jbc     bx,1,prntsp               ; Print space instead of 0
  256A 38280C                  134           jbs     divisor,0,chr_ok          ; If in rightmost position print 0
  256D 3A0015            E     135   prntsp: jbs     bx,2,cont         ; Do not print space if bit is set
  2570 A1F00024                136           ld      value,#0F0H       ;  0F0h+30h = 20H = space
  2574 2003                    137           br      chr_ok
                               138
  2576 910100            E     139   non_0:  orb     bx,#0001B                 ; Set flag so 0's will be printed
  2579 65300024                140   chr_ok: add     value,#30h                ; 30h + n = 0 to 9 ascii
  257D 617F0024                141           and     value,#7Fh        ; send least sig seven bits, clear upper word
  2581 C824                    142           push    value
  2583 2F8B                    143           call    con_out           ; output ascii result (result<9)
  2585 A02624                  144   cont:   ld      value,value+2     ; load value with remainder
  2588 012A                    145           clr     divisor+2
  258A 8D0A0028                146           divu    divisor,#10       ; next lower power of ten
  258E 880028            E     147           cmp     divisor,zero
  2591 D7C7                    148           jne     div_loop
  2593                         149   div_done:
  2593 E300              E     150           br      [cx]
                               151
  2596                         152   DIVTAB:         ;       Number of places for result
  2596 000001000A006400        153           dcw     0, 1, 10, 100, 1000, 10000      ; divisor table - 10**n
```

270189-67

```
MCS-96 MACRO ASSEMBLER    PLOT_SERIAL                          02/18/86          PAGE    5

ERR LOC  OBJECT                LINE       SOURCE STATEMENT
                               154
                               155
                               156
                               157
    25A2                       158   DRAW_GRAPH:                     ; Graph drawing routine
    25A2 C90D00                159         push    #0dh
    25A5 2F69                  160         call    con_out
    25A7 C90A82                161         push    #820AH            ;;;     Clear 3 lines
    25AA 2F64                  162         call    CON_OUT
    25AC C90000                163         push    #00
    25AF 2F5F                  164         call    CON_out
                               165
    25B1 012C                  166         clr     xptr
    25B3 0130                  167         clr     xval
    25B5                       168   NXT_ROW:
    25B5 C90D0A                169         push    #0A0DH            ; CRLF
    25B8 2F56                  170         call    CON_OUT
    25BA C90000                171         push    #00H              ; nul
    25BD 2F51                  172         call    CON_OUT
                               173
    25BF C830                  174         push    xval
    25C1 C9020A                175         push    #(0A00H or 0010b)       ; supress all zeros except rightmost
    25C4 2F86                  176         call    PRINT_NUM
                               177
    25C6 C92020                178         push    #2020H            ; Print 2 spaces
    25C9 2F45                  179         call    CON_OUT
    25CB C92B00                180         push    #2BH              ; +
    25CE 2F40                  181         call    con_out
                               182
    25D0 A100002E          E   183         ld      yptr,#PLOT_RES_2        ; PLOT_RES_2 = PLOT_RES/2
                               184                                         ; PLOT_RES is defined 7 lines down
                               185
    25D4                       186   NXT_COL:                              ; Next Column
    25D4 8B2D00002E        E   187         cmp     yptr,PLOT_IN[xptr]
    25D9 D911                  188         jh      PRT_NUM
    25DB                       189   PRT_MK:                               ; Print Mark
    25DB C92A00                190         push    #2AH
    25DE 2F30                  191         call    CON_OUT
    25E0                       192   INC_CNT:
    25E0 6500002E          E   193         add     yptr,#PLOT_RES  ; PLOT_RES = number of inputs per output point
    25E4 8900002E          E   194         cmp     yptr,#PLOT_MAX  ; PLOT_max = maximum line length
    25E8 D1EA                  195         jnh     nxt_col
    25EA 204F                  196         br      NXTLN
                               197   $eject
```

270189-68

Listing 5—The Plot Module (Continued)

```
MCS-96 MACRO ASSEMBLER    PLOT_SERIAL                                  02/18/86          PAGE    6

ERR LOC  OBJECT                LINE       SOURCE STATEMENT
                               198
    25EC                       199   PRT_NUM:
    25EC 8900002E          E   200           cmp     yptr,#PLOT_RES_2        ; If value is less then minimum needed
    25F0 DF49                  201           be      NXTLN                   ; for a plot, do not print value
                               202
    25F2 C92020                203           push    #2020H                  ; print 2 spaces then value
    25F5 2F19                  204           call    con_out
    25F7 3B000B            E   205           JBS     FFT_MODE,3,db_mode
                               206
    25FA                       207   norm_mode:
    25FA CB2D0000          E   208           push    PLOT_IN[xptr]
    25FE C9000A                209           push    #(0A00H or 0000B)       ; supress all zeros
    2601 2F49                  210           call    PRINT_NUM
    2603 2036                  211           BR      NXTLN
                               212
    2605                       213   db_mode:
    2605 A32D00002E        E   214           ld      yptr,plot_in[xptr]      ; PLOT_IN = 512*10*LOG(x)
    260A 08012E                215           shr     yptr,#1                 ; yptr=265 * 10LOG(x)
    260D AC2F00            E   216           ldbze   ax,yptr+1               ; ax= 10LOG(x) = yptr/256
                               217
    2610 C800              E   218           push    ax                      ; Print AX
    2612 C9020A                219           push    #(0A00H or 0010B)       ; supress all but rightmost zero
    2615 2F35                  220           call    PRINT_NUM
    2617 C92E00                221           push    #2EH                    ; Decimal point
    261A 2EF4                  222           call    con_out
                               223
    261C B02E01            E   224           ldb     ax+1,yptr       ; high byte of ax = fractional portion of
    261F 1100              E   225           clrb    ax              ;                   10LOG(x)
                               226
    2621 6DE60300          E   227           mulu    ax,#3E6H        ; if ax=FF00H then ax+2 now = 998 decimal
    2625 370102            E   228           jbc     ax+1,7,no_rnd
    2628 0700              E   229           inc     dx              ; round value up
                               230
    262A C800              E   231   no_rnd: push    dx      ; dx=ax+2
    262C C90106                232           push    #(600H or 0001B)        ; print all numbers to three places
    262F 2F1B                  233           call    Print_num
    2631 C92000                234           push    #20H            ; space
    2634 2EDA                  235           call    con_out
    2636 C96442                236           push    #4264H          ; "dB"
    2639 2ED5                  237           call    con_out
                               238
                               239   $eject
```

270189-69

```
MCS-96 MACRO ASSEMBLER    PLOT_SERIAL                          02/18/86        PAGE    7

ERR LOC  OBJECT              LINE       SOURCE STATEMENT
                              240
                              241                                ; Setup for next line
    263B C90D0A               242   NXTLN:  push    #0A0DH       ; CRLF
    263E 2ED0                 243           call    CON_OUT
    2640 C90000               244           push    #00H         ; nul
    2643 2ECB                 245           call    CON_OUT
    2645 C92086               246           push    #8620H       ; 7 spaces
    2648 2EC6                 247           call    CON_OUT
    264A C92100               248           push    #21H         ; !
    264D 2EC1                 249           call    con_out
                              250
    264F 0730                 251           inc     xval
    2651 6502002C             252           add     xptr,#2
    2655 893E002C             253           cmp     xptr,#62
    2659 D2022758             254   !       ble     nxt_row      ; Start printing next row
                              255
    265D C90D0A               256   Done:   push    #0A0DH       ; CRLF  ; Form feed for next graph
    2660 2EAE                 257           call    CON_OUT
    2662 C9000C               258           push    #0C00H       ; null,FF
    2665 2EA9                 259           call    con_out
                              260
    2667 F0                   261           RET
    2668                      262   END

ASSEMBLY COMPLETED,   NO ERROR(S) FOUND.
```

270189-70

Listing 5—The Plot Module (Continued)

At the end of the plot, a form feed is given to set the printer up for the next graph. Our printer would frequently miss the character after a CRLF. To solve this problem, a null (ASCII 0) is sent after every CRLF to make sure the printer is ready for the next line. This has been found to be a problem with many devices running at close to their maximum capacity, and the nulls work well to solve it.

With the plot completed, the program begins to run again by taking another set of A to D samples.

11.0 USING THE FFT PROGRAM

The program can be used with either real or tabled data. If real data is used, the signal is applied to analog channel 1. The program as written performs A/D samples at 100 microsecond intervals, collecting the 64 samples in 6.4 milliseconds. This sets the sampling window frequency at 156 Hz. If tabled data is used, 64 words of data should be placed in the location pointed to by DATA0 in the TABLE_LOAD routine of the Main Module.

Program control is specified by FFT_MODE which is loaded in the main module. Also within the main module are settings which control the A to D buffer routine and the Plot routine. The intention was to have only one module to change and recompile to vary parameters in the entire program.

The program modules are set up to run one-at-a-time so that the code would be easy to understand. Additionally, the Plot routine takes so long relative to the other sections, that it doesn't pay to try to overlap code sections. If this code were to be converted to run a process instead of print a graph, it might be worthwhile to run the FFT and the A/D routines at the same time.

If the goal of a modified program is to have the highest frequency sampling possible, it might be desirable to streamline the A/D section and run it without interruption. When the A to D routine was complete the FFT routine could be started. The reasoning behind this is that at the fastest A/D speeds the processor will be almost completely tied up processing the A/D information and storing it away. Using an interrupt based A/D routine would slow things down.

A set of programs which will perform a FFT has been presented in this application note. These programs are available from the INSITE users library as program CA-26. More importantly, dozens of programing examples have been made available, making it easier to get started with the 8096. Examples of how to use the hardware on the 8096 have already appeared in AP-248, "Using The 8096". These two applications notes form a good base for the understanding of MCS-96 microcontroller based design.

12.0 APPENDIX A - MATRICES

Matrices are a convenient way to express groups of equations. Consider the complex discrete Fourier Transform in equation 9, with N = 4.

$$Y_n = \sum_{k=0}^{3} X(k)\, W^{nk} \qquad n = 0, 1, 2, 3$$

This can be expanded to

$$\begin{aligned}
Y(0) &= X(0)\,W^0 + X(1)\,W^0 + X(2)\,W^0 + X(3)\,W^0 \\
Y(1) &= X(0)\,W^0 + X(1)\,W^1 + X(2)\,W^2 + X(3)\,W^3 \\
Y(2) &= X(0)\,W^0 + X(1)\,W^2 + X(2)\,W^4 + X(3)\,W^6 \\
Y(3) &= X(0)\,W^0 + X(1)\,W^3 + X(2)\,W^6 + X(3)\,W^9
\end{aligned}$$

In matrix notation, this is shown as

$$\begin{bmatrix} Y(0) \\ Y(1) \\ Y(2) \\ Y(3) \end{bmatrix} = \begin{bmatrix} W^0 & W^0 & W^0 & W^0 \\ W^0 & W^1 & W^2 & W^3 \\ W^0 & W^2 & W^4 & W^6 \\ W^0 & W^3 & W^6 & W^9 \end{bmatrix} \begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \end{bmatrix}$$

The first step to simplifying this is to reduce the center matrix. Recalling that

$$W^N = W^{N \text{ MOD } N} \quad \text{and} \quad W^0 = 1$$

The matrix can be reduced to have less non-trivial multiplications.

$$\begin{bmatrix} Y(0) \\ Y(1) \\ Y(2) \\ Y(3) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & W^1 & W^2 & W^3 \\ 1 & W^2 & W^0 & W^2 \\ 1 & W^3 & W^2 & W^1 \end{bmatrix} \begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \end{bmatrix}$$

The square matrix can be factored into

$$\begin{bmatrix} Y(0) \\ Y(2) \\ Y(1) \\ Y(3) \end{bmatrix} = \begin{bmatrix} 1 & W^0 & 0 & 0 \\ 1 & W^2 & 0 & 0 \\ 0 & 0 & 1 & W^1 \\ 0 & 0 & 1 & W^3 \end{bmatrix} \begin{bmatrix} 1 & 0 & W^0 & 0 \\ 0 & 1 & 0 & W^0 \\ 1 & 0 & W^2 & 0 \\ 0 & 1 & 0 & W^2 \end{bmatrix} \begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \end{bmatrix}$$

For this equation to work, the Y(1) and Y(2) terms need to be swapped, as shown above. This procedure is a Bit Reversal, as described in the text.

Multiplying the two rightmost matrices results in

$$\begin{aligned}
&X(0) + X(2)\,W^0 \\
&X(1) + X(3)\,W^0 \\
&X(0) + X(2)\,W^3 \\
&X(1) + X(3)\,W^2
\end{aligned}$$

requiring 4 complex multiplications & 4 complex additions

Noting that $W^0 = -W^2$, 2 of the complex multiplications can be eliminated, with the following results

$$\begin{aligned}
&X(0) + X(2)\,W^0 \\
&X(1) + X(3)\,W^0 \\
&X(0) - X(2)\,W^0 \\
&X(1) - X(3)\,W^0
\end{aligned}$$

requiring 2 complex multiplications and 4 complex additions

Since $W^1 = -W^3$, a similar result occurs when this vector is multiplied by the remaining square matrix. The resulting equations are:

$$\begin{aligned}
Y(0) &= (X(0) + X(2)\,W^0) + W^0\,(X(0) + X(3)\,W^0) \\
Y(2) &= (X(0) + X(2)\,W^0) - W^0\,(X(1) + X(3)\,W^0) \\
Y(1) &= (X(0) - X(2)\,W^0) + W^1\,(X(1) - X(3)\,W^0) \\
Y(3) &= (X(0) - X(2)\,W^0) - W^1\,(X(1) - X(3)\,W^0)
\end{aligned}$$

The number of complex multiplications required is 4, as compared with 16 for the unfactored matrix.

In general, the FFT requires

$$\frac{N * \text{EXPONENT}}{2} \text{ complex multiplications}$$

and

$$N * \text{EXPONENT complex additions}$$

where

$$\text{EXPONENT} = \text{Log}_2 N$$

A standard Fourier Transform requires

$$N^2 \text{ complex multiplications}$$

and

$$N(N-1) \text{ complex additions}$$

13.0 APPENDIX B - PLOTS

The following plots are examples of output from the FFT program. These plots were generated using tabled data, but very similar plots have also been made using the analog input module. Typically, a plot made using the analog input module will not show quite as much power at each frequency and will show a positive value for the DC component. This is because it is difficult to get exactly a full-scale analog input with no DC offset.

Plot 1 is a Magnitude plot of a square wave of period NT.

Plot 2 is the same data plotted in dB. Note how the dB plot enhances the difference in the small signal values at the high frequencies.

Plot 3 shows the windowed version of this data. Note that the widening of the bins due to windowing shows energy in the even harmonics that is not actually present. For data of this type a different window other than Hanning would normally be used. Many window types are available, the selection of which can be determined by the type of data to be plotted.[3]

Plot 4 shows a sine wave of period NT/7 or frequency 7/NT.

Plot 5 shows the same input with windowing. Note the signal shown in bins 6 and 8.

Plot 6 shows a sine wave of period NT/7.5. Note the noise caused by the discontinuity as discussed earlier.

Plot 7 uses windowing on the data used for plot 6. Note the cleaner appearance.

Plot 8 shows a sine wave input of magnitude 0.707 and period NT/7.5.

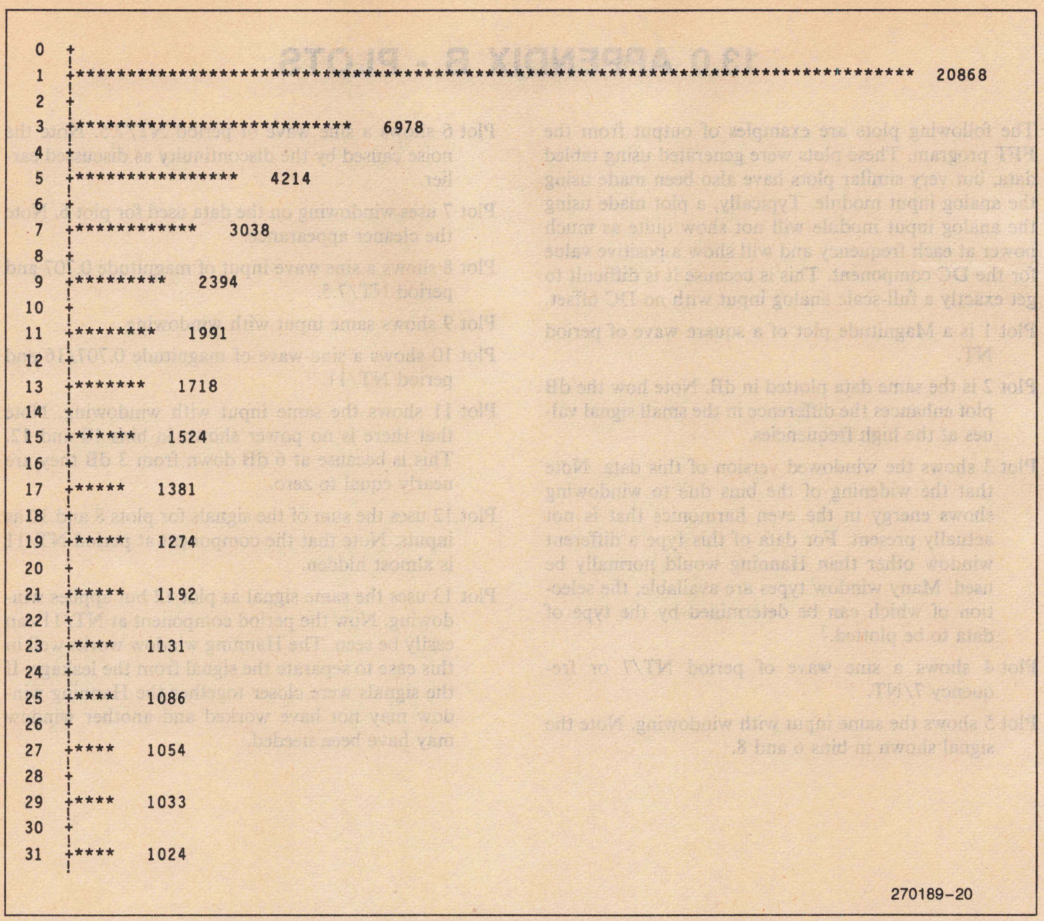
Plot 9 shows same input with windowing.

Plot 10 shows a sine wave of magnitude 0.707/16 and period NT/11.

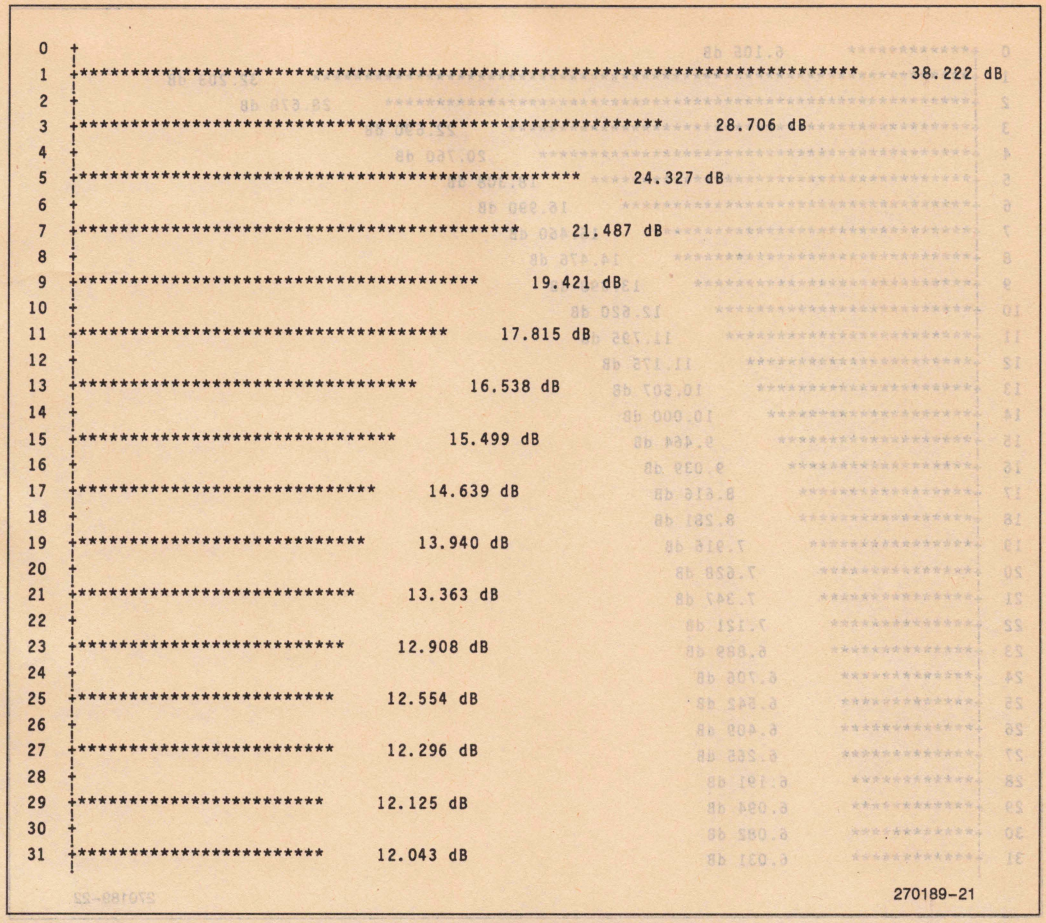
Plot 11 shows the same input with windowing. Note that there is no power shown in bins 10 and 12. This is because at 6 dB down from 3 dB they are nearly equal to zero.

Plot 12 uses the sum of the signals for plots 8 and 10 as inputs. Note that the component at period NT/11 is almost hidden.

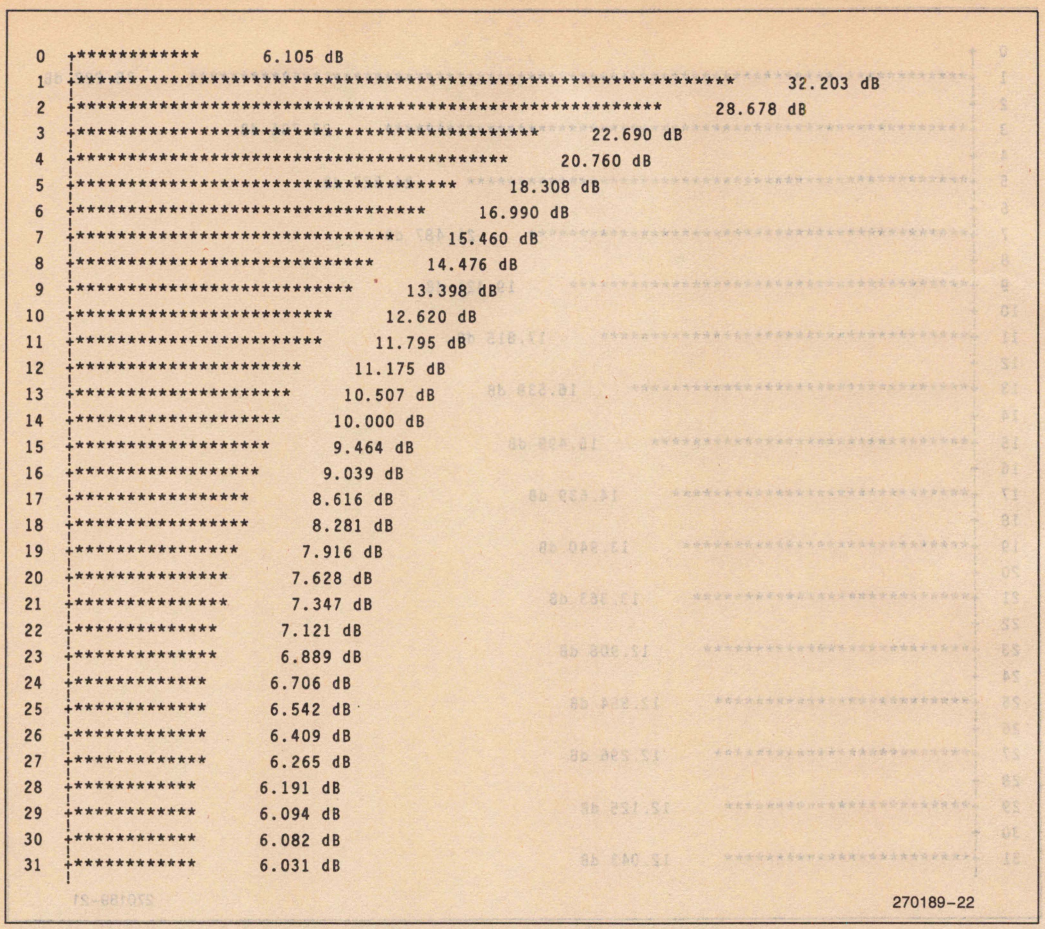
Plot 13 uses the same signal as plot 12 but applies windowing. Now the period component at NT/11 can easily be seen. The Hanning window works well in this case to separate the signal from the leakage. If the signals were closer together the Hanning window may not have worked and another window may have been needed.



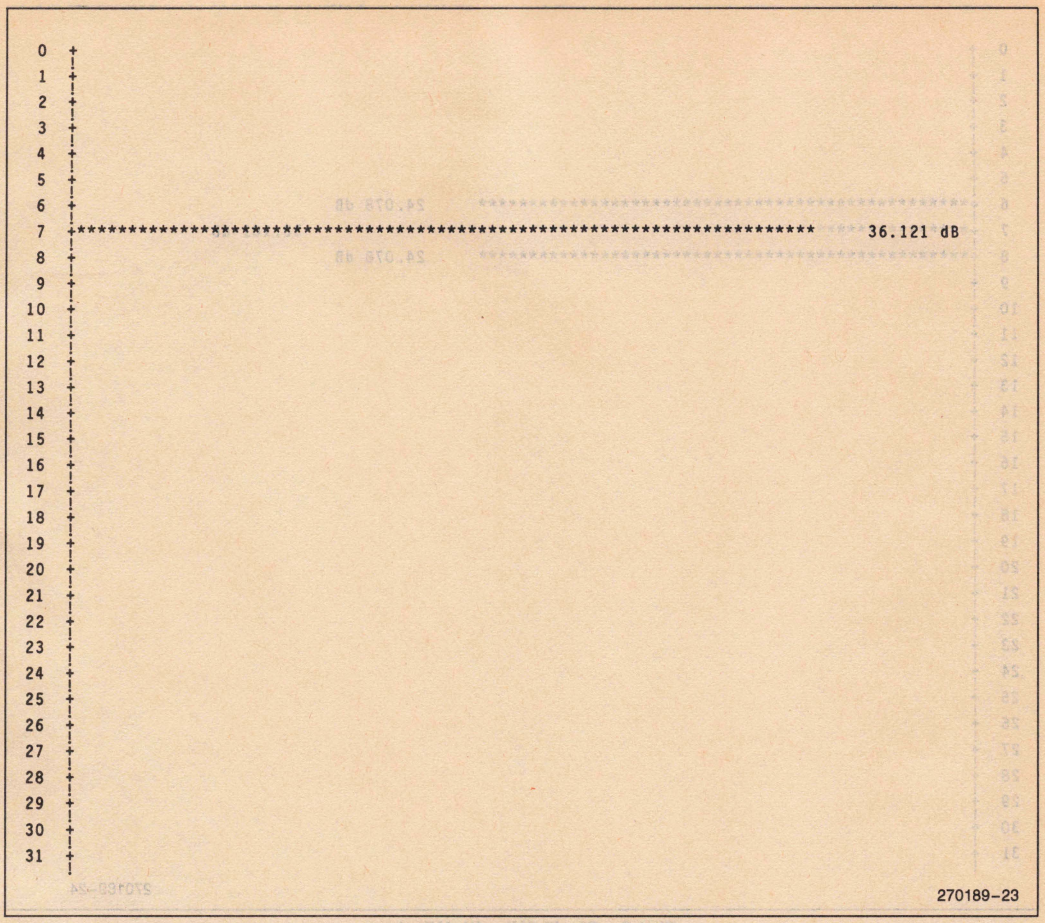
Plot 1—Magnitude Plot of Squarewave



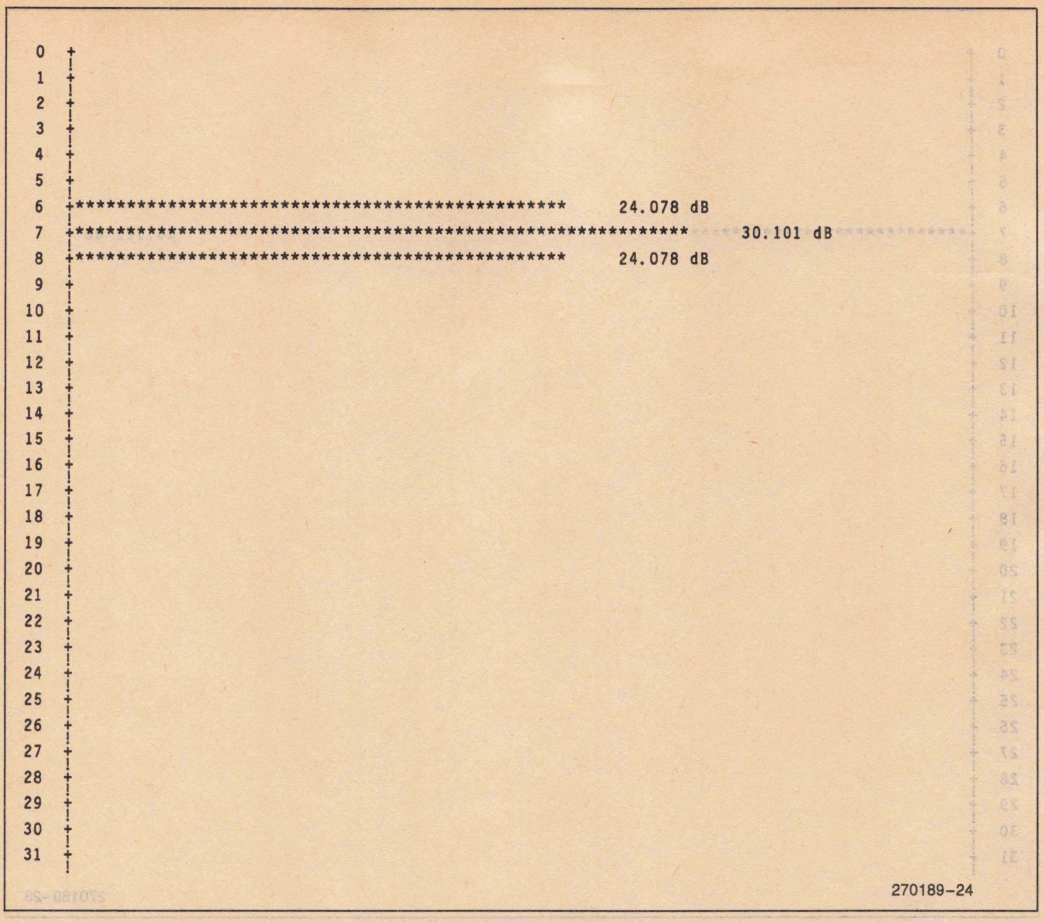
Plot 2—Decibel Plot of Squarewave



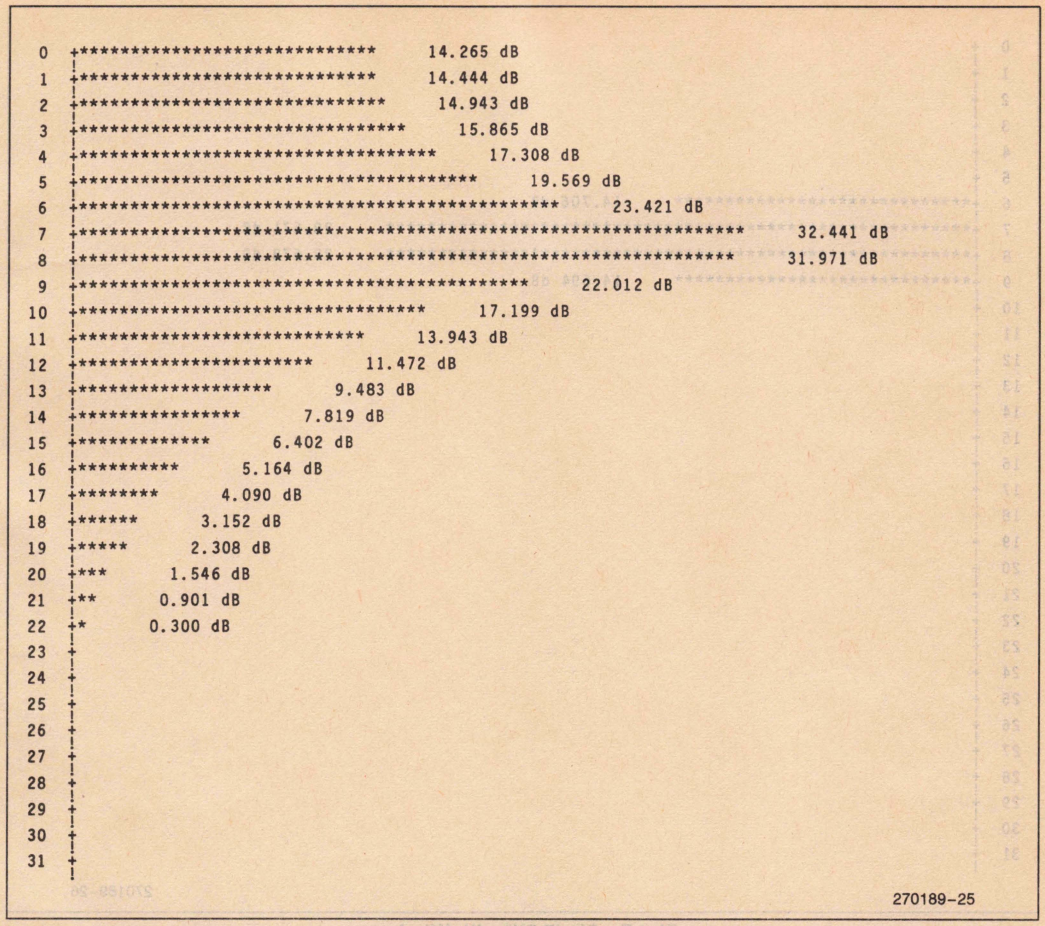
Plot 3—Plot of Squarewave with Window



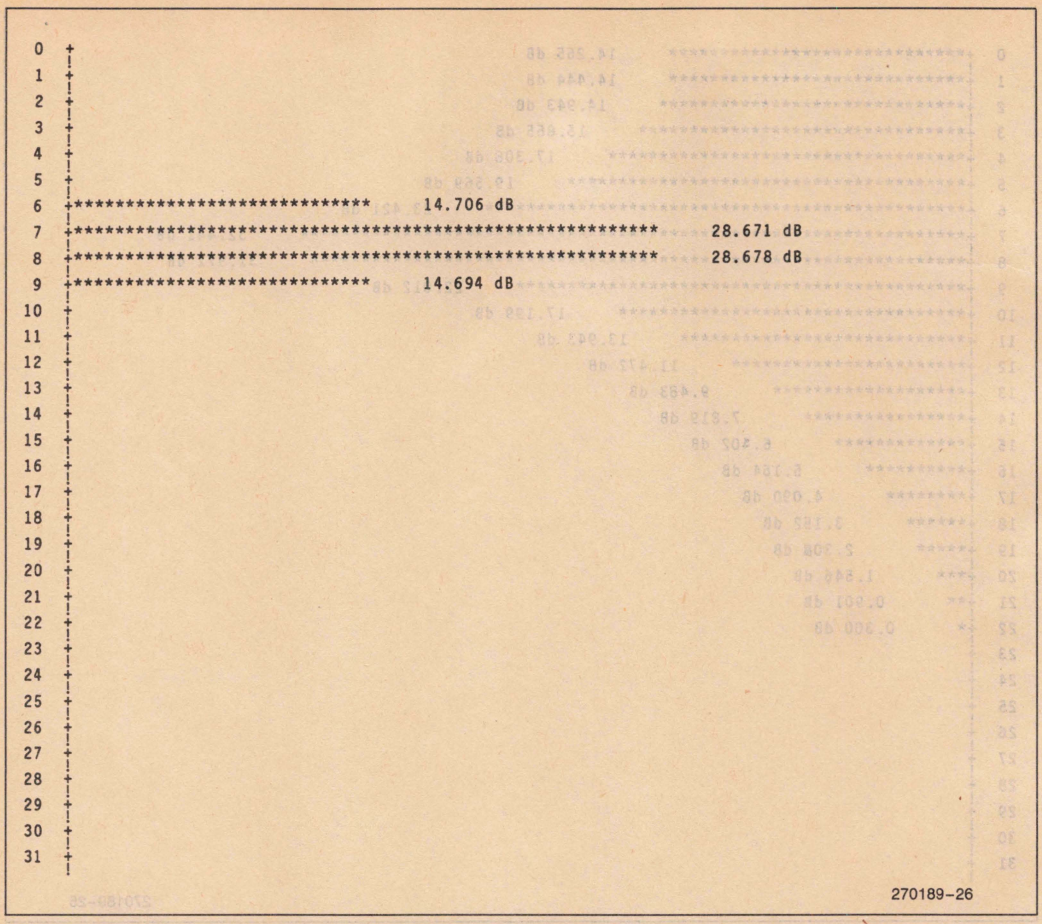
Plot 4—Sin (7.0X) without Window



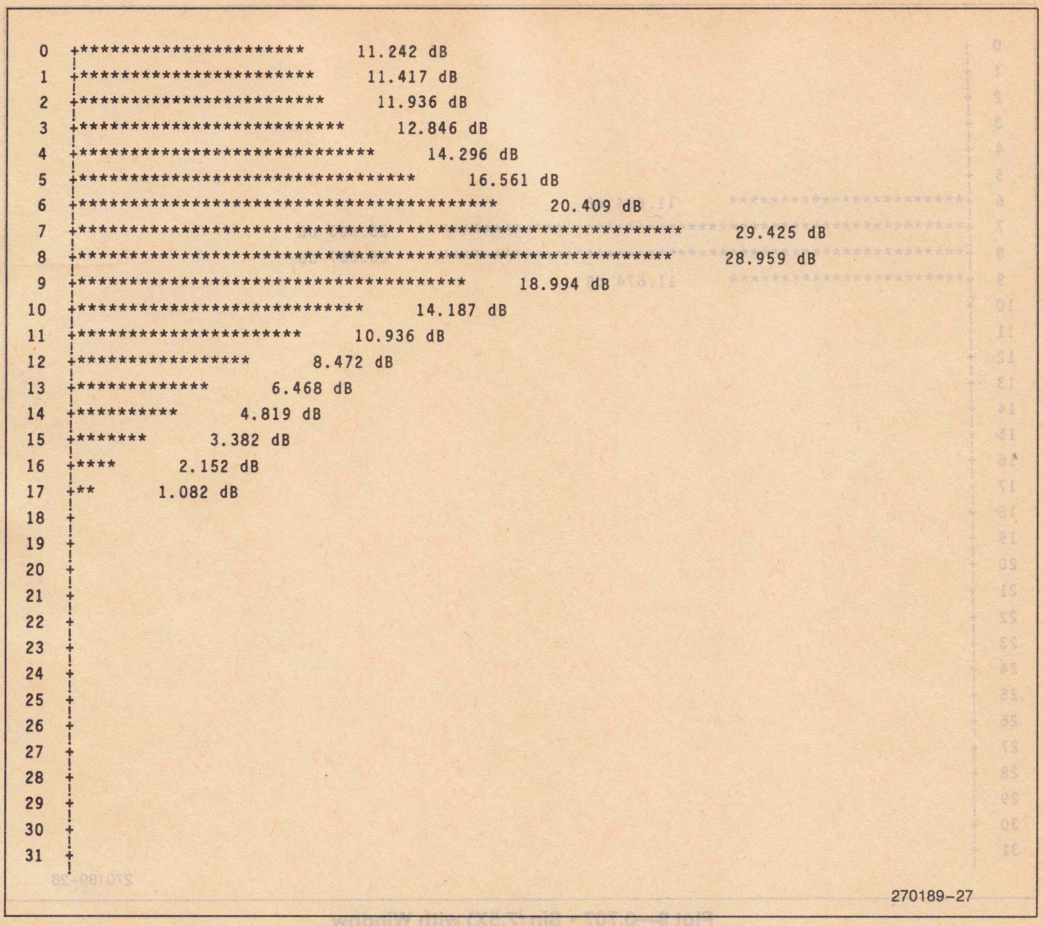
Plot 5—Sin (7.0X) with Window



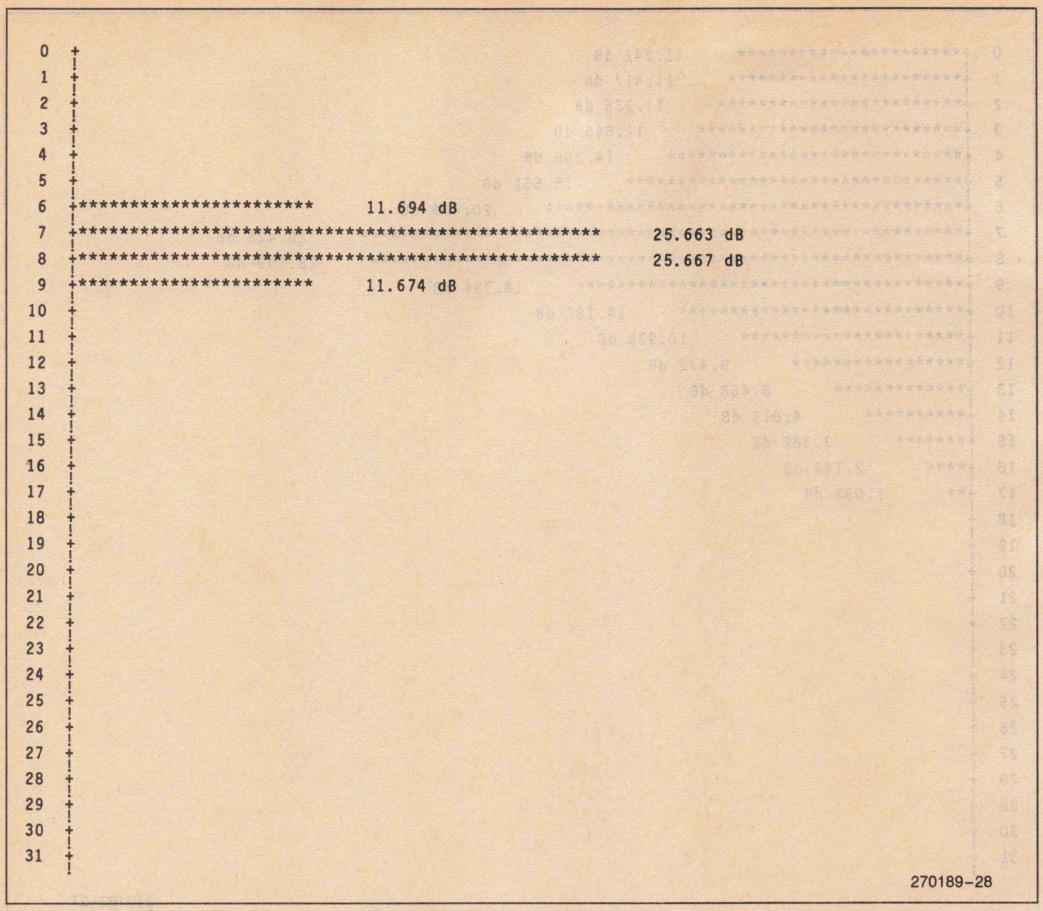
Plot 6—Sin (7.5X) without Window



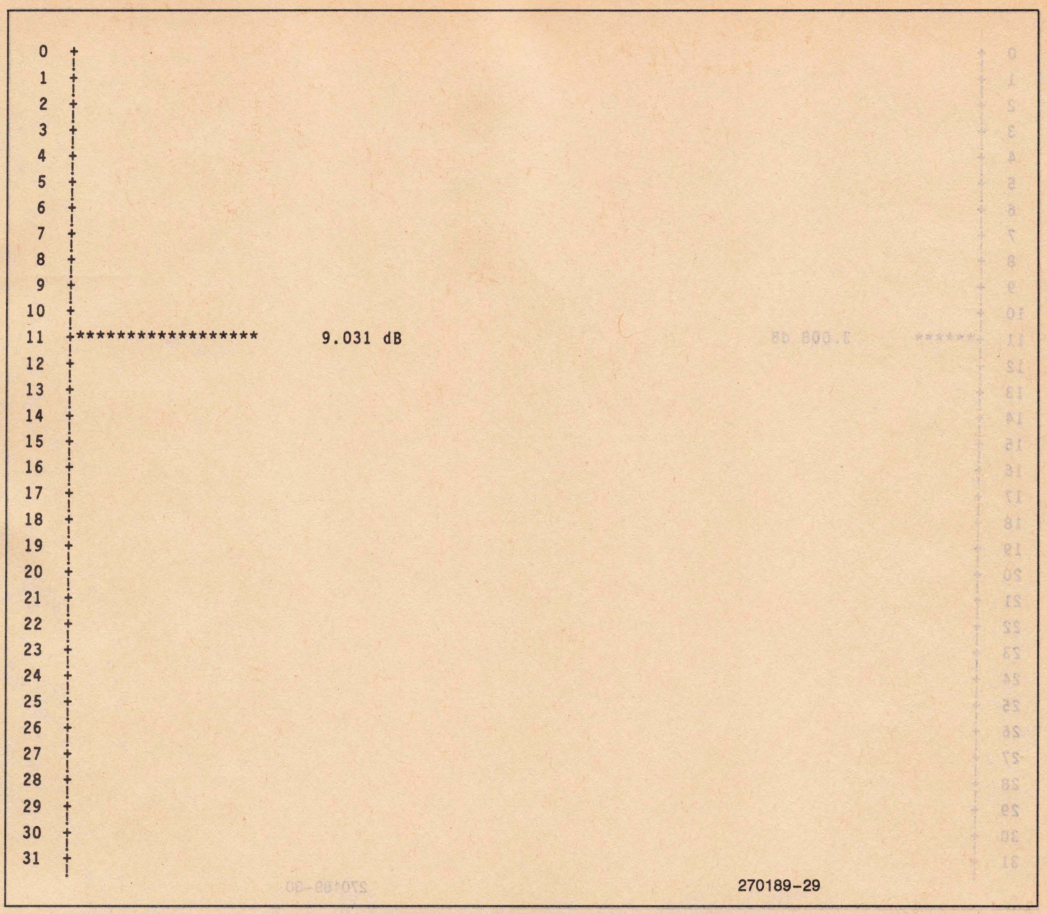
Plot 7—Sin (7.5X) with Window



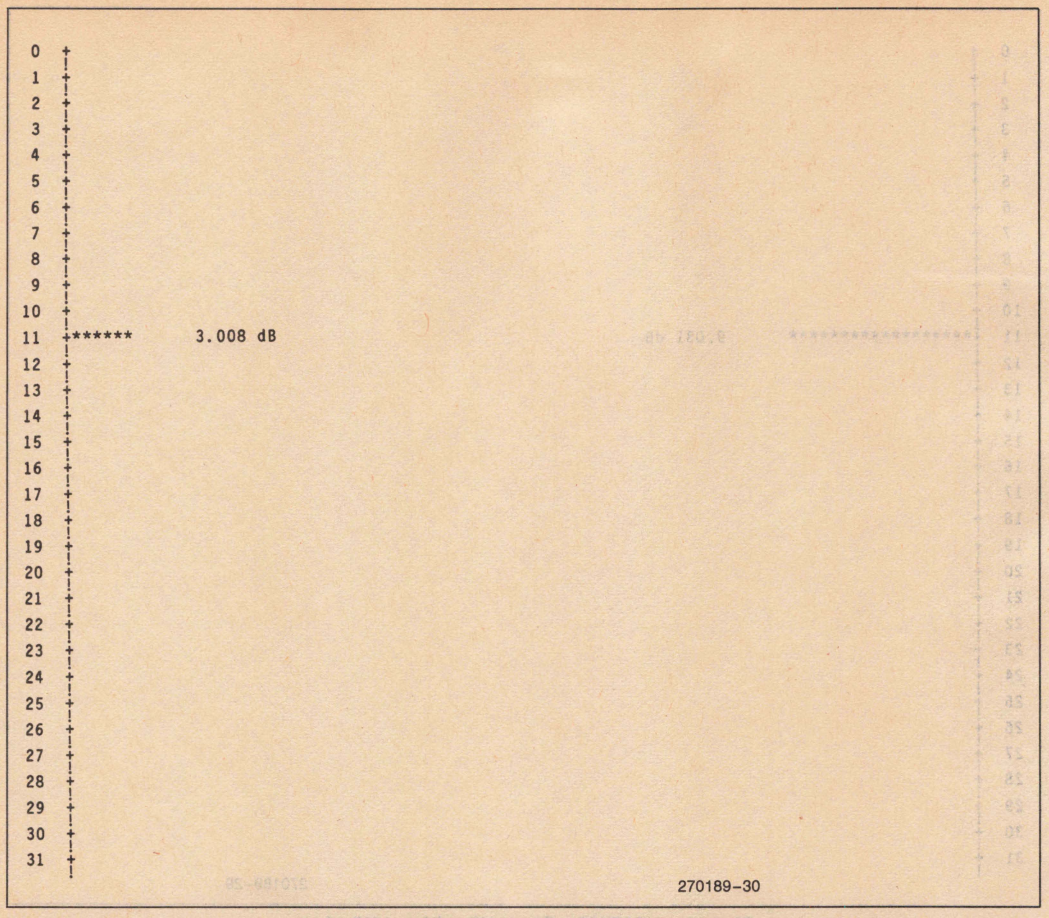
Plot 8—0.707 * Sin (7.5X) without Window



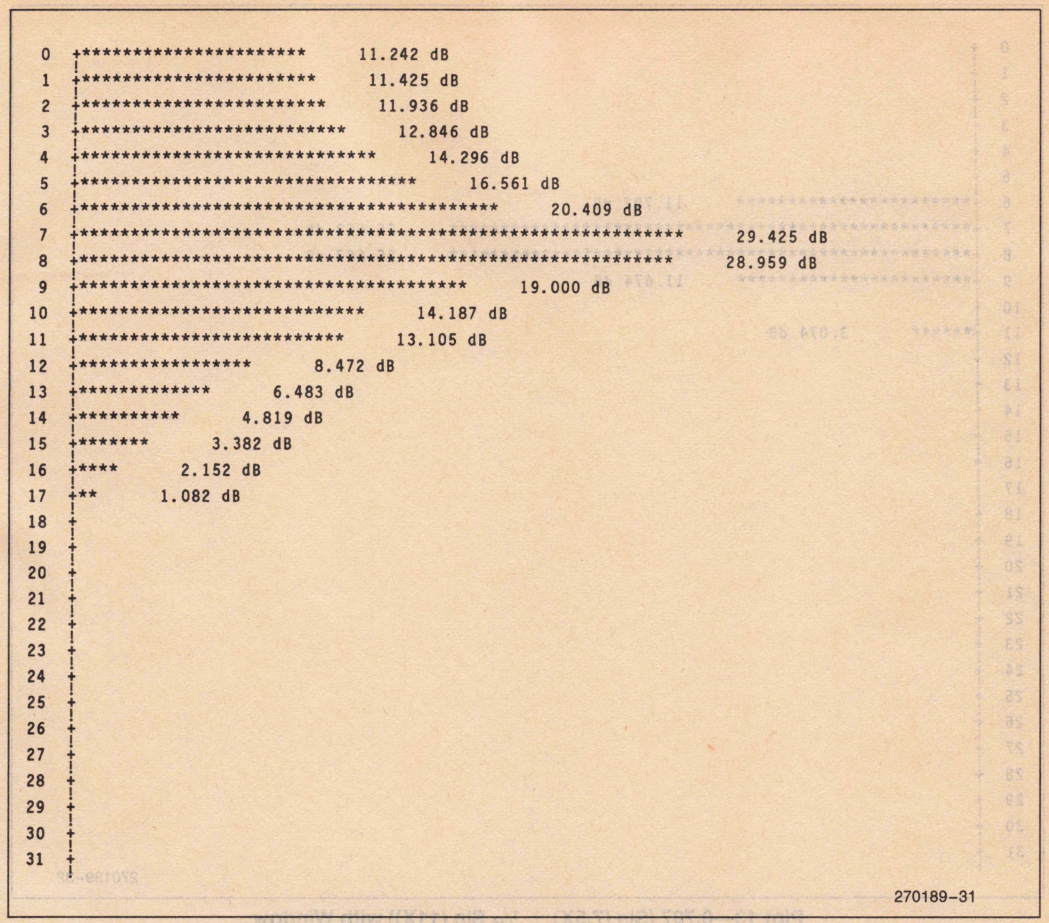
Plot 9—0.707 * Sin (7.5X) with Window



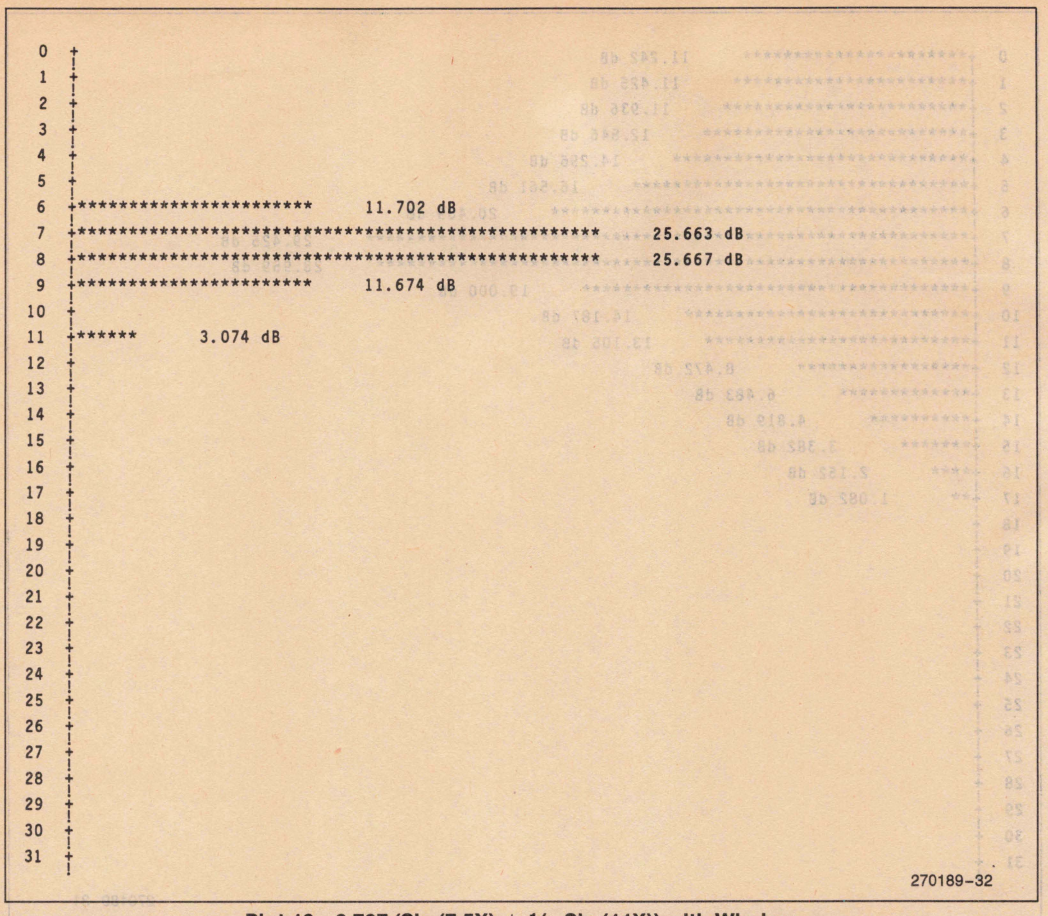
Plot 10—0.707/16 * Sin (11X) without Window



Plot 11—0.707/16 * Sin (11X) with Window



Plot 12—0.707 (Sin (7.5X) + 1/16 Sin (11X)) without Window



Plot 13—0.707 (Sin (7.5X) + 1/16 Sin (11X)) with Window

BIBLIOGRAPHY

1. Boyet, Howard and Katz, Ron, The 16-Bit 8096: Programming, Interfacing, Applications. 1985, Microprocessor Training Inc., New York, NY.
2. Brigham, E. Oran, The Fast Fourier Transform. 1974, Prentice-Hall, Inc., Englewood Cliffs, New Jersey.
3. Harris, Fredric J., On the use of Windows for Harmonic Analysis with the Discrete Fourier Transform. Proceedings of the IEEE, Vol. 66, No. 1, January 1978.
4. Weaver, H. Joseph, Applications of discrete and continuous Fourier analysis. 1983, John Wiley and Sons, New York.

INTEL PUBLICATIONS

1. 1986 Microcontroller Handbook, Order Number 210918-004
2. Using the 8096, AP-248, Order Number 270061-001
3. MCS-96 Macro Assembler User's Guide, Order Number 122048-001
4. MCS-96 Utilities User's Guide, Order Number 122049-001

BIBLIOGRAPHY

1. Bovet, Howard and Katz, Ron, The 16-Bit 8096: Programming, Interfacing, Applications, 1985, Microprocessor Training Inc., New York, NY.

2. Brigham, E. Oran, The Fast Fourier Transform, 1974, Prentice-Hall, Inc., Englewood Cliffs, New Jersey.

3. Harris, Fredric J., On the use of Windows for Harmonic Analysis with the Discrete Fourier Transform, Proceedings of the IEEE, Vol. 66, No. 1, January 1978.

4. Weaver, H. Joseph, Applications of discrete and continuous Fourier analysis, 1983, John Wiley and Sons, New York.

INTEL PUBLICATIONS

1. 1986 Microcontroller Handbook, Order Number 210918-004

2. Using the 8096, AP-248, Order Number 270081-001

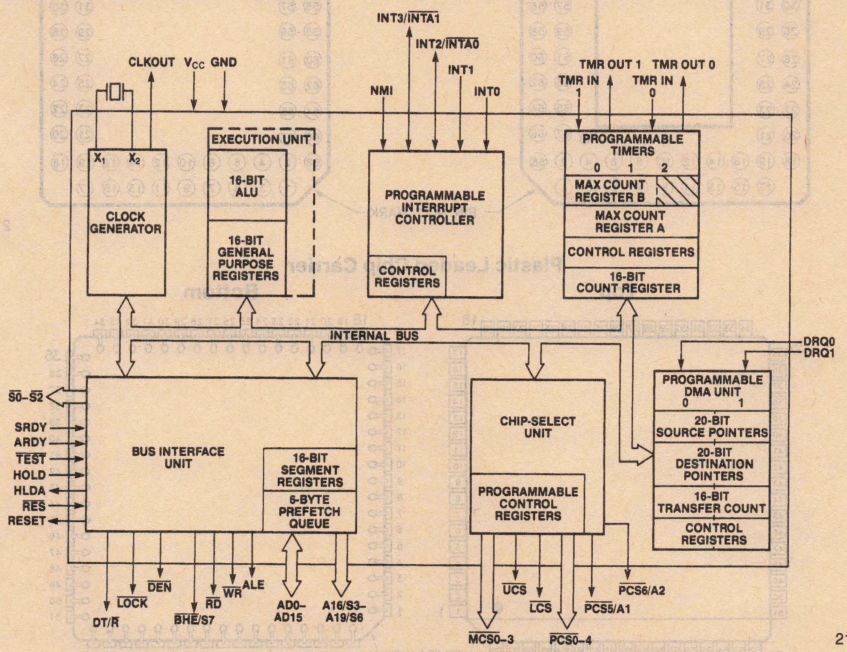
3. MCS-96 Macro Assembler User's Guide, Order Number 122048-001

4. MCS-96 Utilities User's Guide, Order Number 122049-001

80186
HIGH INTEGRATION 16-BIT MICROPROCESSOR

- **Integrated Feature Set**
  - — Enhanced 8086-2 CPU
  - — Clock Generator
  - — 2 Independent, High-Speed DMA Channels
  - — Programmable Interrupt Controller
  - — 3 Programmable 16-bit Timers
  - — Programmable Memory and Peripheral Chip-Select Logic
  - — Programmable Wait State Generator
  - — Local Bus Controller
- **Available in 12.5 MHz (80186-12), 10 MHz (80186-10) and 8 MHz (80186) Versions**
- **High-Performance Processor**
  - — At 8 MHz provides 2 times the Performance of the Standard 8086
  - — 4 MByte/Sec Bus Bandwidth Interface @ 8 MHz
  - — 6.25 MByte/Sec Bus Bandwidth Interface @ 12.5 MHz
- **Direct Addressing Capability to 1 MByte of Memory and 64 KByte I/O**
- **Completely Object Code Compatible with All Existing 8086, 8088 Software**
  - — 10 New Instruction Types
- **Complete System Development Support**
  - — Development Software; Assembler, PL/M, Pascal, Fortan, and System Utilities
  - — In-Circuit-Emulator (I^2ICE^{TM}-186)
- **High Performance Numerical Coprocessing Capability Through 8087 Interface**
- **Available in 68 Pin:**
  - — Plastic Leaded Chip Carrier (PLCC)
  - — Ceramic Pin Grid Array (PGA)
  - — Ceramic Leadless Chip Carrier (LCC)

  (See Packaging Spec, Order #231369)
- **Available in EXPRESS**
  - — Standard Temperature with Burn-In
  - — Extended Temperature Range (−40°C to +85°C)



210451-1

Figure 1. 80186 Block Diagram

October 1986
Order Number: 210451-007

The Intel 80186 is a highly integrated 16-bit microprocessor. The 80186 effectively combines 15–20 of the most common 8086 system components onto one. The 80186 provides two times greater throughput than the standard 5 MHz 8086. The 80186 is upward compatible with 8086 and 8088 software and adds 10 new instruction types to the existing set.

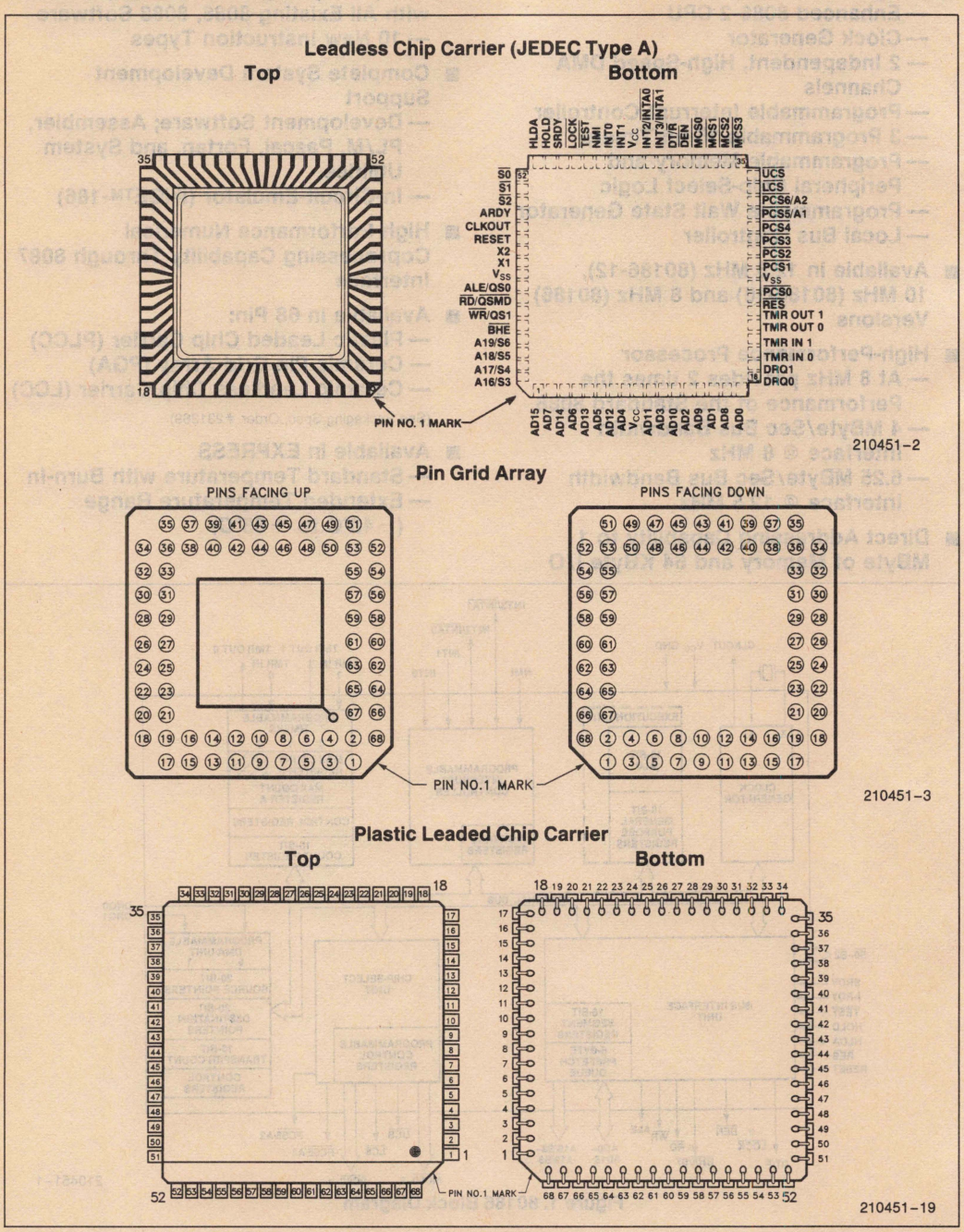


Figure 2. 80186 Pinout Diagrams

Table 1. 80186 Pin Description

Symbol	Pin No.	Type	Name and Function
V_{CC}, V_{CC}	9, 43	I	**System Power:** +5 volt power supply.
V_{SS}, V_{SS}	26, 60	I	System Ground.
RESET	57	O	Reset Output indicates that the 80186 CPU is being reset, and can be used as a system reset. It is active HIGH, synchronized with the processor clock, and lasts an integer number of clock periods corresponding to the length of the $\overline{RES}$ signal.
X1, X2	59, 58	I	Crystal Inputs, X1 and X2, provide an external connection for a fundamental mode parallel resonant crystal for the internal crystal oscillator. X1 can interface to an external clock instead of a crystal. In this case, minimize the capacitance on X2 or drive X2 with complemented X1. The input or oscillator frequency is internally divided by two to generate the clock signal (CLKOUT).
CLKOUT	56	O	Clock Output provides the system with a 50% duty cycle waveform. All device pin timings are specified relative to CLKOUT. CLKOUT has sufficient MOS drive capabilities for the 8087 Numeric Processor Extension.
$\overline{RES}$	24	I	System Reset causes the 80186 to immediately terminate its present activity, clear the internal logic, and enter a dormant state. This signal may be asynchronous to the 80186 clock. The 80186 begins fetching instructions approximately 7 clock cycles after $\overline{RES}$ is returned HIGH. $\overline{RES}$ is required to be LOW for greater than 4 clock cycles and is internally synchronized. For proper initialization, the LOW-to-HIGH transition of $\overline{RES}$ must occur no sooner than 50 microseconds after power up. This input is provided with a Schmitt-trigger to facilitate power-on $\overline{RES}$ generation via an RC network. When $\overline{RES}$ occurs, the 80186 will drive the status lines to an inactive level for one clock, and then tri-state them.
$\overline{TEST}$	47	I	$\overline{TEST}$ is examined by the WAIT instruction. If the $\overline{TEST}$ input is HIGH when "WAIT" execution begins, instruction execution will suspend. $\overline{TEST}$ will be resampled until it goes LOW, at which time execution will resume. If interrupts are enabled while the 80186 is waiting for $\overline{TEST}$, interrupts will be serviced. This input is synchronized internally.
TMR IN 0, TMR IN 1	20 21	I I	Timer Inputs are used either as clock or control signals, depending upon the programmed timer mode. These inputs are active HIGH (or LOW-to-HIGH transitions are counted) and internally synchronized.
TMR OUT 0, TMR OUT 1	22 23	O O	Timer outputs are used to provide single pulse or continous waveform generation, depending upon the timer mode selected.
DRQ0 DRQ1	18 19	I I	DMA Request is driven HIGH by an external device when it desires that a DMA channel (Channel 0 or 1) perform a transfer. These signals are active HIGH, level-triggered, and internally synchronized.
NMI	46	I	Non-Maskable Interrupt is an edge-triggered input which causes a type 2 interrupt. NMI is not maskable internally. A transition from a LOW to HIGH initiates the interrupt at the next instruction boundary. NMI is latched internally. An NMI duration of one clock or more will guarantee service. This input is internally synchronized.
INT0, INT1 INT2/$\overline{INTA0}$ INT3/$\overline{INTA1}$	45, 44 42 41	I I/O I/O	Maskable Interrupt Requests can be requested by strobing one of these pins. When configured as inputs, these pins are active HIGH. Interrupt Requests are synchronized internally. INT2 and INT3 may be configured via software to provide active-LOW interrupt-acknowledge output signals. All interrupt inputs may be configured via software to be either edge- or level-triggered. To ensure recognition, all interrupt requests must remain active until the interrupt is acknowledged. When iRMX mode is selected, the function of these pins changes (see Interrupt Controller section of this data sheet).

Table 1. 80186 Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
A19/S6, A18/S5, A17/S4, A16/S3	65 66 67 68	O O O O	Address Bus Outputs (16–19) and Bus Cycle Status (3–6) reflect the four most significant address bits during T_1. These signals are active HIGH. During T_2, T_3, T_W, and T_4, status information is available on these lines as encoded below:
			Low / **High**
			S6: Processor Cycle / DMA Cycle
			S3, S4, and S5 are defined as LOW during T_2–T_4.
AD15–AD0	10–17, 1–8	I/O	Address/Data Bus (0–15) signals constitute the time multiplexed memory or I/O address (T_1) and data (T_2, T_3, T_W, and T_4) bus. The bus is active HIGH. A_0 is analogous to $\overline{BHE}$ for the lower byte of the data bus, pins D_7 through D_0. It is LOW during T_1 when a byte is to be transferred onto the lower portion of the bus in memory or I/O operations.
$\overline{BHE}$/S7	64	O	During T_1 the Bus High Enable signal should be used to determine if data is to be enabled onto the most significant half of the data bus; pins D_{15}–D_8. $\overline{BHE}$ is LOW during T_1 for read, write, and interrupt acknowledge cycles when a byte is to be transferred on the higher half of the bus. The S_7 status information is available during T_2, T_3, and T_4. S_7 is logically equivalent to $\overline{BHE}$. The signal is active LOW, and is tristated OFF during bus HOLD.
			$\overline{BHE}$ and A0 Encodings
			$\overline{BHE}$ Value / **A0 Value** / **Function**
			0 / 0 / Word Transfer
			0 / 1 / Byte Transfer on upper half of data bus (D15–D8)
			1 / 0 / Byte Transfer on lower half of data bus (D_7–D_0)
			1 / 1 / Reserved
ALE/QS0	61	O	Address Latch Enable/Queue Status 0 is provided by the 80186 to latch the address into the 8282/8283 address latches. ALE is active HIGH. Addresses are guaranteed to be valid on the trailing edge of ALE. The ALE rising edge is generated off the rising edge of the CLKOUT immediately preceding T_1 of the associated bus cycle, effectively one-half clock cycle earlier than in the standard 8086. The trailing edge is generated off the CLKOUT rising edge in T_1 as in the 8086. Note that ALE is never floated.
$\overline{WR}$/QS1	63	O	Write Strobe/Queue Status 1 indicates that the data on the bus is to be written into a memory or an I/O device. $\overline{WR}$ is active for T_2, T_3, and T_W of any write cycle. It is active LOW, and floats during "HOLD." It is driven HIGH for one clock during Reset, and then floated. When the 80186 is in queue status mode, the ALE/QS0 and $\overline{WR}$/QS1 pins provide information about processor/instruction queue interaction.
			QS1 / **QS0** / **Queue Operation**
			0 / 0 / No queue operation
			0 / 1 / First opcode byte fetched from the queue
			1 / 1 / Subsequent byte fetched from the queue
			1 / 0 / Empty the queue

Table 1. 80186 Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
$\overline{RD}/\overline{QSMD}$	62	O	Read Strobe indicates that the 80186 is performing a memory or I/O read cycle. $\overline{RD}$ is active LOW for T_2, T_3, and T_W of any read cycle. It is guaranteed not to go LOW in T_2 until after the Address Bus is floated. $\overline{RD}$ is active LOW, and floats during "HOLD". $\overline{RD}$ is driven HIGH for one clock during Reset, and then the output driver is floated. A weak internal pull-up mechanism of the $\overline{RD}$ line holds it HIGH when the line is not driven. During RESET the pin is sampled to determine whether the 80186 should provide ALE, $\overline{WR}$ and $\overline{RD}$, or if the Queue-Status should be provided. $\overline{RD}$ should be connected to GND to provide Queue-Status data.
ARDY	55	I	Asynchronous Ready informs the 80186 that the addressed memory space or I/O device will complete a data transfer. The ARDY input pin will accept an asynchronous input, and is active HIGH. Only the rising edge is internally synchronized by the 80186. This means that the falling edge of ARDY must be synchronized to the 80186 clock. If connected to V_{CC}, no WAIT states are inserted. Asynchronous ready (ARDY) or synchronous ready (SRDY) must be active to terminate a bus cycle. If unused, this line should be tied LOW.
SRDY	49	I	Synchronous Ready must be synchronized externally to the 80186. The use of SRDY provides a relaxed system-timing specification on the Ready input. This is accomplished by eliminating the one-half clock cycle which is required for internally resolving the signal level when using the ARDY input. This line is active HIGH. If this line is connected to V_{CC}, no WAIT states are inserted. Asynchronous ready (ARDY) or synchronous ready (SRDY) must be active before a bus cycle is terminated. If unused, this line should be tied LOW.
$\overline{LOCK}$	48	O	$\overline{LOCK}$ output indicates that other system bus masters are not to gain control of the system bus while $\overline{LOCK}$ is active LOW. The $\overline{LOCK}$ signal is requested by the LOCK prefix instruction and is activated at the beginning of the first data cycle associated with the instruction following the LOCK prefix. It remains active until the completion of the instruction following the LOCK prefix. No prefetches will occur while $\overline{LOCK}$ is asserted. When executing more than one LOCK instruction, always make sure there are 6 bytes of code between the end of the first LOCK instruction and the start of the second LOCK instruction. $\overline{LOCK}$ is active LOW, is driven HIGH for one clock during RESET, and then floated.
$\overline{S0}$, $\overline{S1}$, $\overline{S2}$	52–54	O	Bus cycle status $\overline{S0}$–$\overline{S2}$ are encoded to provide bus-transaction information:

80186 Bus Cycle Status Information

$\overline{S2}$	$\overline{S1}$	$\overline{S0}$	Bus Cycle Initiated
0	0	0	Interrupt Acknowledge
0	0	1	Read I/O
0	1	0	Write I/O
0	1	1	Halt
1	0	0	Instruction Fetch
1	0	1	Read Data from Memory
1	1	0	Write Data to Memory
1	1	1	Passive (no bus cycle)

The status pins float during "HOLD."
$\overline{S2}$ may be used as a logical M/$\overline{IO}$ indicator, and $\overline{S1}$ as a DT/$\overline{R}$ indicator.
The status lines are driven HIGH for one clock during Reset, and then floated until a bus cycle begins.

Table 1. 80186 Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
HOLD (input) HLDA (output)	50 51	I O	HOLD indicates that another bus master is requesting the local bus. The HOLD input is active HIGH. HOLD may be asynchronous with respect to the 80186 clock. The 80186 will issue a HLDA (HIGH) in response to a HOLD request at the end of T_4 or T_1. Simultaneous with the issuance of HLDA, the 80186 will float the local bus and control lines. After HOLD is detected as being LOW, the 80186 will lower HLDA. When the 80186 needs to run another bus cycle, it will again drive the local bus and control lines.
$\overline{UCS}$	34	O	Upper Memory Chip Select is an active LOW output whenever a memory reference is made to the defined upper portion (1K–256K block) of memory. This line is not floated during bus HOLD. The address range activating $\overline{UCS}$ is software programmable.
$\overline{LCS}$	33	O	Lower Memory Chip Select is active LOW whenever a memory reference is made to the defined lower portion (1K–256K) of memory. This line is not floated during bus HOLD. The address range activating $\overline{LCS}$ is software programmable.
$\overline{MCS0}$–3	38, 37, 36, 35	O	Mid-Range Memory Chip Select signals are active LOW when a memory reference is made to the defined mid-range portion of memory (8K–512K). These lines are not floated during bus HOLD. The address ranges activating $\overline{MCS0}$–3 are software programmable.
$\overline{PCS0}$ $\overline{PCS1}$–4	25 27, 28, 29, 30	O O	Peripheral Chip Select signals 0–4 are active LOW when a reference is made to the defined peripheral area (64K byte I/O space). These lines are not floated during bus HOLD. The address ranges activating $\overline{PCS0}$–4 are software programmable.
$\overline{PCS5}$/A1	31	O	Peripheral Chip Select 5 or Latched. A1 may be programmed to provide a sixth peripheral chip select, or to provide an internally latched A1 signal. The address range activating $\overline{PCS5}$ is software programmable. When programmed to provide latched. A1, rather than $\overline{PCS5}$, this pin will retain the previously latched value of A1 during a bus HOLD. A1 is active HIGH.
$\overline{PCS6}$/A2	32	O	Peripheral Chip Select 6 or Latched A2 may be programmed to provide a seventh peripheral chip select, or to provide an internally latched A2 signal. The address range activating $\overline{PCS6}$ is software programmable. When programmed to provide latched A2, rather than $\overline{PCS6}$, this pin will retain the previously latched value of A2 during a bus HOLD. A2 is active HIGH.
DT/$\overline{R}$	40	O	Data Transmit/Receive controls the direction of data flow through the external 8286/8287 data bus transceiver. When LOW, data is transferred to the 80186. When HIGH the 80186 places write data on the data bus.
$\overline{DEN}$	39	O	Data Enable is provided as an 8286/8287 data bus transceiver output enable. $\overline{DEN}$ is active LOW during each memory and I/O access. $\overline{DEN}$ is HIGH whenever DT/$\overline{R}$ changes state.

FUNCTIONAL DESCRIPTION

Introduction

The following Functional Description describes the base architecture of the 80186. This architecture is common to the 8086, 8088, and 80286 microprocessor families as well. The 80186 is a very high integration 16-bit microprocessor. It combines 15–20 of the most common microprocessor system components onto one chip while providing twice the performance of the standard 8086. The 80186 is object code compatible with the 8086/8088 microprocessors and adds 10 new instruction types to the existing 8086/8088 instruction set.

80186 BASE ARCHITECTURE

The 8086, 8088, 80186, and 80286 family all contain the same basic set of registers, instructions, and addressing modes. The 80186 processor is upward compatible with the 8086, 8088, and 80286 CPUs.

Register Set

The 80186 base architecture has fourteen registers as shown in Figures 3a and 3b. These registers are grouped into the following categories.

General Registers

Eight 16-bit general purpose registers may be used to contain arithmetic and logical operands. Four of these (AX, BX, CX, and DX) can be used as 16-bit registers or split into pairs of separate 8-bit registers.

Segment Registers

Four 16-bit special purpose registers select, at any given time, the segments of memory that are immediately addressable for code, stack, and data. (For usage, refer to Memory Organization.)

Base and Index Registers

Four of the general purpose registers may also be used to determine offset addresses of operands in memory. These registers may contain base addresses or indexes to particular locations within a segment. The addressing mode selects the specific registers for operand and address calculations.

Status and Control Registers

Two 16-bit special purpose registers record or alter certain aspects of the 80186 processor state. These are the Instruction Pointer Register, which contains the offset address of the next sequential instruction to be executed, and the Status Word Register, which contains status and control flag bits (see Figures 3a and 3b).

Status Word Description

The Status Word records specific characteristics of the result of logical and arithmetic instructions (bits 0, 2, 4, 6, 7, and 11) and controls the operation of the 80186 within a given operating mode (bits 8, 9, and 10). The Status Word Register is 16-bits wide. The function of the Status Word bits is shown in Table 2.

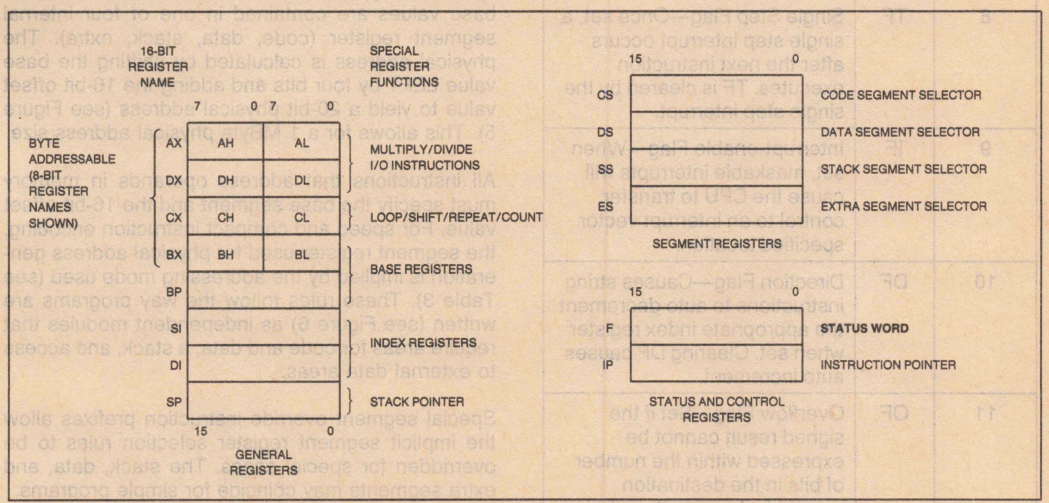


Figure 3a. 80186 General Purpose Register Set

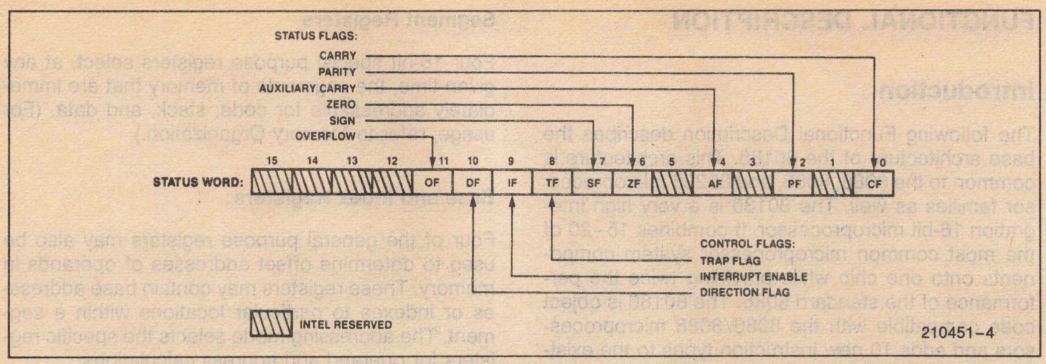


Figure 3b. Status Word Format

Table 2. Status Word Bit Functions

Bit Position	Name	Function
0	CF	Carry Flag—Set on high-order bit carry or borrow; cleared otherwise
2	PF	Parity Flag—Set if low-order 8 bits of result contain an even number of 1-bits; cleared otherwise
4	AF	Set on carry from or borrow to the low order four bits of AL; cleared otherwise
6	ZF	Zero Flag—Set if result is zero; cleared otherwise
7	SF	Sign Flag—Set equal to high-order bit of result (0 if positive, 1 if negative)
8	TF	Single Step Flag—Once set, a single step interrupt occurs after the next instruction executes. TF is cleared by the single step interrupt.
9	IF	Interrupt-enable Flag—When set, maskable interrupts will cause the CPU to transfer control to an interrupt vector specified location.
10	DF	Direction Flag—Causes string instructions to auto decrement the appropriate index register when set. Clearing DF causes auto increment.
11	OF	Overflow Flag—Set if the signed result cannot be expressed within the number of bits in the destination operand; cleared otherwise

Instruction Set

The instruction set is divided into seven categories: data transfer, arithmetic, shift/rotate/logical, string manipulation, control transfer, high-level instructions, and processor control. These categories are summarized in Figure 4.

An 80186 instruction can reference anywhere from zero to several operands. An operand can reside in a register, in the instruction itself, or in memory. Specific operand addressing modes are discussed later in this data sheet.

Memory Organization

Memory is organized in sets of segments. Each segment is a linear contiguous sequence of up to 64K (2^{16}) 8-bit bytes. Memory is addressed using a two-component address (a pointer) that consists of a 16-bit base segment and a 16-bit offset. The 16-bit base values are contained in one of four internal segment register (code, data, stack, extra). The physical address is calculated by shifting the base value LEFT by four bits and adding the 16-bit offset value to yield a 20-bit physical address (see Figure 5). This allows for a 1 MByte physical address size.

All instructions that address operands in memory must specify the base segment and the 16-bit offset value. For speed and compact instruction encoding, the segment register used for physical address generation is implied by the addressing mode used (see Table 3). These rules follow the way programs are written (see Figure 6) as independent modules that require areas for code and data, a stack, and access to external data areas.

Special segment override instruction prefixes allow the implicit segment register selection rules to be overridden for special cases. The stack, data, and extra segments may coincide for simple programs.

GENERAL PURPOSE	
MOV	Move byte or word
PUSH	Push word onto stack
POP	Pop word off stack
PUSHA	Push all registers on stack
POPA	Pop all registers from stack
XCHG	Exchange byte or word
XLAT	Translate byte
INPUT/OUTPUT	
IN	Input byte or word
OUT	Output byte or word
ADDRESS OBJECT	
LEA	Load effective address
LDS	Load pointer using DS
LES	Load pointer using ES
FLAG TRANSFER	
LAHF	Load AH register from flags
SAHF	Store AH register in flags
PUSHF	Push flags onto stack
POPF	Pop flags off stack
ADDITION	
ADD	Add byte or word
ADC	Add byte or word with carry
INC	Increment byte or word by 1
AAA	ASCII adjust for addition
DAA	Decimal adjust for addition
SUBTRACTION	
SUB	Subtract byte or word
SBB	Subtract byte or word with borrow
DEC	Decrement byte or word by 1
NEG	Negate byte or word
CMP	Compare byte or word
AAS	ASCII adjust for subtraction
DAS	Decimal adjust for subtraction
MULTIPLICATION	
MUL	Multiply byte or word unsigned
IMUL	Integer multiply byte or word
AAM	ASCII adjust for multiply
DIVISION	
DIV	Divide byte or word unsigned
IDIV	Integer divide byte or word
AAD	ASCII adjust for division
CBW	Convert byte to word
CWD	Convert word to doubleword

MOVS	Move byte or word string
INS	Input bytes or word string
OUTS	Output bytes or word string
CMPS	Compare byte or word string
SCAS	Scan byte or word string
LODS	Load byte or word string
STOS	Store byte or word string
REP	Repeat
REPE/REPZ	Repeat while equal/zero
REPNE/REPNZ	Repeat while not equal/not zero
LOGICALS	
NOT	"Not" byte or word
AND	"And" byte or word
OR	"Inclusive or" byte or word
XOR	"Exclusive or" byte or word
TEST	"Test" byte or word
SHIFTS	
SHL/SAL	Shift logical/arithmetic left byte or word
SHR	Shift logical right byte or word
SAR	Shift arithmetic right byte or word
ROTATES	
ROL	Rotate left byte or word
ROR	Rotate right byte or word
RCL	Rotate through carry left byte or word
RCR	Rotate through carry right byte or word
FLAG OPERATIONS	
STC	Set carry flag
CLC	Clear carry flag
CMC	Complement carry flag
STD	Set direction flag
CLD	Clear direction flag
STI	Set interrupt enable flag
CLI	Clear interrupt enable flag
EXTERNAL SYNCHRONIZATION	
HLT	Halt until interrupt or reset
WAIT	Wait for $\overline{\text{TEST}}$ pin active
ESC	Escape to extension processor
LOCK	Lock bus during next instruction
NO OPERATION	
NOP	No operation
HIGH LEVEL INSTRUCTIONS	
ENTER	Format stack for procedure entry
LEAVE	Restore stack for procedure exit
BOUND	Detects values outside prescribed range

Figure 4. 80186 Instruction Set

CONDITIONAL TRANSFERS	
JA/JNBE	Jump if above/not below nor equal
JAE/JNB	Jump if above or equal/not below
JB/JNAE	Jump if below/not above nor equal
JBE/JNA	Jump if below or equal/not above
JC	Jump if carry
JE/JZ	Jump if equal/zero
JG/JNLE	Jump if greater/not less nor equal
JGE/JNL	Jump if greater or equal/not less
JL/JNGE	Jump if less/not greater nor equal
JLE/JNG	Jump if less or equal/not greater
JNC	Jump if not carry
JNE/JNZ	Jump if not equal/not zero
JNO	Jump if not overflow
JNP/JPO	Jump if not parity/parity odd
JNS	Jump if not sign

JO	Jump if overflow
JP/JPE	Jump if parity/parity even
JS	Jump if sign
UNCONDITIONAL TRANSFERS	
CALL	Call procedure
RET	Return from procedure
JMP	Jump
ITERATION CONTROLS	
LOOP	Loop
LOOPE/LOOPZ	Loop if equal/zero
LOOPNE/LOOPNZ	Loop if not equal/not zero
JCXZ	Jump if register CX = 0
INTERRUPTS	
INT	Interrupt
INTO	Interrupt if overflow
IRET	Interrupt return

Figure 4. 80186 Instruction Set (Continued)

To access operands that do not reside in one of the four immediately available segments, a full 32-bit pointer can be used to reload both the base (segment) and offset values.

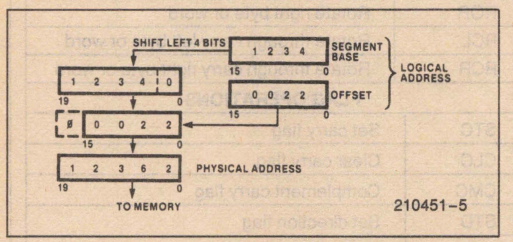


Figure 5. Two Component Address

Table 3. Segment Register Selection Rules

Memory Reference Needed	Segment Register Used	Implicit Segment Selection Rule
Instructions	Code (CS)	Instruction prefetch and immediate data.
Stack	Stack (SS)	All stack pushes and pops; any memory references which use BP Register as a base register.
External Data (Global)	Extra (ES)	All string instruction references which use the DI register as an index.
Local Data	Data (DS)	All other data references.

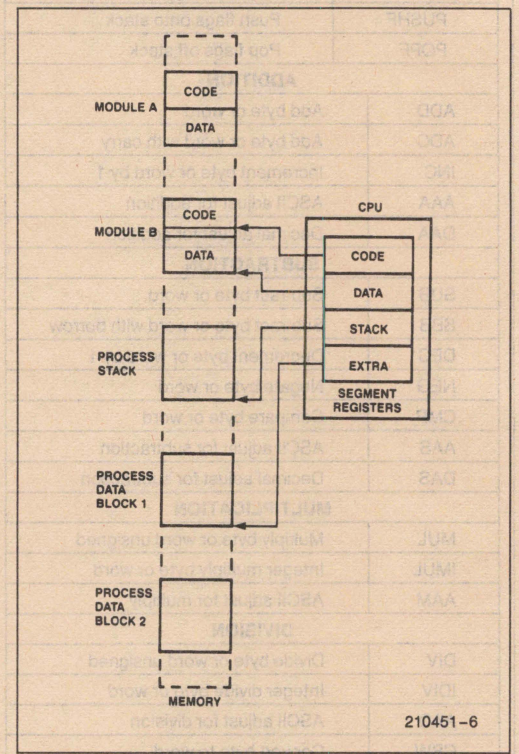


Figure 6. Segmented Memory Helps Structure Software

Addressing Modes

The 80186 provides eight categories of addressing modes to specify operands. Two addressing modes are provided for instructions that operate on register or immediate operands:

- *Register Operand Mode:* The operand is located in one of the 8- or 16-bit general registers.
- *Immediate Operand Mode:* The operand is included in the instruction.

Six modes are provided to specify the location of an operand in a memory segment. A memory operand address consists of two 16-bit components: a segment base and an offset. The segment base is supplied by a 16-bit segment register either implicitly chosen by the addressing mode or explicitly chosen by a segment override prefix. The offset, also called the effective address, is calculated by summing any combination of the following three address elements:

- the *displacement* (an 8- or 16-bit immediate value contained in the instruction);
- the *base* (contents of either the BX or BP base registers); and
- the *index* (contents of either the SI or DI index registers).

Any carry out from the 16-bit addition is ignored. Eight-bit displacements are sign extended to 16-bit values.

Combinations of these three address elements define the six memory addressing modes, described below.

- *Direct Mode:* The operand's offset is contained in the instruction as an 8- or 16-bit displacement element.
- *Register Indirect Mode:* The operand's offset is in one of the registers SI, DI, BX, or BP.
- *Based Mode:* The operand's offset is the sum of an 8- or 16-bit displacement and the contents of a base register (BX or BP).
- *Indexed Mode:* The operand's offset is the sum of an 8- or 16-bit displacement and the contents of an index register (SI or DI).
- *Based Indexed Mode:* The operand's offset is the sum of the contents of a base register and an Index register.
- *Based indexed Mode with Displacement:* The operand's offset is the sum of a base register's contents, an index register's contents, and an 8- or 16-bit displacement.

Data Types

The 80186 directly supports the following data types:

- *Integer:* A signed binary numeric value contained in an 8-bit byte or a 16-bit word. All operations assume a 2's complement representation. Signed 32- and 64-bit integers are supported using the 80186/20 Numeric Data Processor.
- *Ordinal:* An unsigned binary numeric value contained in an 8-bit byte or a 16-bit word.
- *Pointer:* A 16- or 32-bit quantity, composed of a 16-bit offset component or a 16-bit segment base component in addition to a 16-bit offset component.
- *String:* A contiguous sequence of bytes or words. A string may contain from 1 to 64K bytes.
- *ASCII:* A byte representation of alphanumeric and control characters using the ASCII standard of character representation.
- *BCD:* A byte (unpacked) representation of the decimal digits 0–9.
- *Packed BCD:* A byte (packed) representation of two decimal digits (0–9). One digit is stored in each nibble (4-bits) of the byte.
- *Floating Point:* A signed 32-, 64-, or 80-bit real number representation. (Floating point operands are supported using the 80186/20 Numeric Data Processor configuration.)

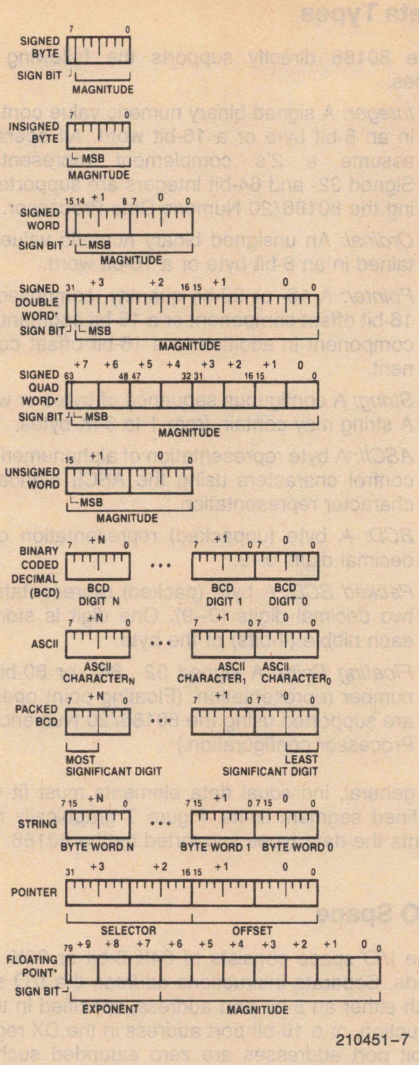
In general, individual data elements must fit within defined segment limits. Figure 7 graphically represents the data types supported by the 80186.

I/O Space

The I/O space consists of 64K 8-bit or 32K 16-bit ports. Separate instructions address the I/O space with either an 8-bit port address, specified in the instruction, or a 16-bit port address in the DX register. 8-bit port addresses are zero extended such that A_{15}–A_8 are LOW. I/O port addresses 00F8(H) through 00FF(H) are reserved.

Interrupts

An interrupt transfers execution to a new program location. The old program address (CS:IP) and machine state (Status Word) are saved on the stack to allow resumption of the interrupted program. Interrupts fall into three classes: hardware initiated, INT instructions, and instruction exceptions. Hardware initiated interrupts occur in response to an external input and are classified as non-maskable or maskable.



NOTE:
*Supported by 80186/20 Numeric Data Processor Configuration.

Figure 7. 80186 Supported Data Types

Programs may cause an interrupt with an INT instruction. Instruction exceptions occur when an unusual condition, which prevents further instruction processing, is detected while attempting to execute an instruction. If the exception was caused by executing an ESC instruction with the ESC trap bit set in the relocation register, the return instruction will point to the ESC instruction, or to the segment override prefix immediately preceding the ESC instruction if the prefix was present. In all other cases, the return address from an exception will point at the instruction immediately following the instruction causing the exception.

A table containing up to 256 pointers defines the proper interrupt service routine for each interrupt. Interrupts 0–31, some of which are used for instruction exceptions, are reserved. Table 4 shows the 80186 predefined types and default priority levels. For each interrupt, an 8-bit vector must be supplied to the 80186 which identifies the appropriate table entry. Exceptions supply the interrupt vector internally. In addition, internal peripherals and noncascaded external interrupts will generate their own vectors through the internal interrupt controller. INT instructions contain or imply the vector and allow access to all 256 interrupts. Maskable hardware initiated interrupts supply the 8-bit vector to the CPU during an interrupt acknowledge bus sequence. Non-maskable hardware interrupts use a predefined internally supplied vector.

Interrupt Sources

The 80186 can service interrupts generated by software or hardware. The software interrupts are generated by specific instructions (INT, ESC, unused OP, etc.) or the results of conditions specified by instructions (array bounds check, INT0, DIV, IDIV, etc.). All interrupt sources are serviced by an indirect call through an element of a vector table. This vector table is indexed by using the interrupt vector type (Table 4), multiplied by four. All hardware-generated interrupts are sampled at the end of each instruction. Thus, the software interrupts will begin service first. Once the service routine is entered and interrupts are enabled, any hardware source of sufficient priority can interrupt the service routine in progress.

The software generated 80186 interrupts are described below.

DIVIDE ERROR EXCEPTION (TYPE 0)

Generated when a DIV or IDIV instruction quotient cannot be expressed in the number of bits in the destination.

SINGLE-STEP INTERRUPT (TYPE 1)

Generated after most instructions if the TF flag is set. Interrupts will not be generated after prefix instructions (e.g., REP), instructions which modify segment registers (e.g., POP DS), or the WAIT instruction.

NON-MASKABLE INTERRUPT—NMI (TYPE 2)

An external interrupt source which cannot be masked.

Table 4. 80186 Interrupt Vectors

Interrupt Name	Vector Type	Default Priority	Related Instructions
Divide Error Exception	0	*1	DIV, IDIV
Single Step Interrupt	1	12**2	All
NMI	2	1	All
Breakpoint Interrupt	3	*1	INT
INT0 Detected Overflow Exception	4	*1	INT0
Array Bounds Exception	5	*1	BOUND
Unused-Opcode Exception	6	*1	Undefined Opcodes
ESC Opcode Exception	7	*1***	ESC Opcodes
Timer 0 Interrupt	8	2A****	
Timer 1 Interrupt	18	2B****	
Timer 2 Interrupt	19	2C****	
Reserved	9	3	
DMA 0 Interrupt	10	4	
DMA 1 Interrupt	11	5	
INT0 Interrupt	12	6	
INT1 Interrupt	13	7	
INT2 Interrupt	14	8	
INT3 Interrupt	15	9	

NOTES:
*1. These are generated as the result of an instruction execution.
**2. This is handled as in the 8086.
****3.All three timers constitute one source of request to the interrupt controller. The Timer interrupts all have the same default priority level with respect to all other interrupt sources. However, they have a defined priority ordering amongst themselves. (Priority 2A is higher priority than 2B.) Each Timer interrupt has a separate vector type number.
4. Default priorities for the interrupt sources are used only if the user does not program each source into a unique priority level.
***5. An escape opcode will cause a trap only if the proper bit is set in the peripheral control block relocation register.

BREAKPOINT INTERRUPT (TYPE 3)

A one-byte version of the INT instruction. It uses 12 as an index into the service routine address table (because it is a type 3 interrupt).

INT0 DETECTED OVERFLOW EXCEPTION (TYPE4)

Generated during an INT0 instruction if the 0F bit is set.

ARRAY BOUNDS EXCEPTION (TYPE 5)

Generated during a BOUND instruction if the array index is outside the array bounds. The array bounds are located in memory at a location indicated by one of the instruction operands. The other operand indicates the value of the index to be checked.

UNUSED OPCODE EXCEPTION (TYPE 6)

Generated if execution is attempted on undefined opcodes.

ESCAPE OPCODE EXCEPTION (TYPE 7)

Generated if execution is attempted of ESC opcodes (D8H–DFH). This exception will only be generated if a bit in the relocation register is set. The return address of this exception will point to the ESC instruction causing the exception. If a segment override prefix preceded the ESC instruction, the return address will point to the segment override prefix.

Hardware-generated interrupts are divided into two groups: maskable interrupts and non-maskable interrupts. The 80186 provides maskable hardware interrupt request pins INT0–INT3. In addition, maskable interrupts may be generated by the 80186 integrated DMA controller and the integrated timer unit. The vector types for these interrupts is shown in Table 4. Software enables these inputs by setting the interrupt flag bit (IF) in the Status Word. The interrupt controller is discussed in the peripheral section of this data sheet.

Further maskable interrupts are disabled while servicing an interrupt because the IF bit is reset as part of the response to an interrupt or exception. The saved Status Word will reflect the enable status of the processor prior to the interrupt. The interrupt flag will remain zero unless specifically set. The interrupt return instruction restores the Status Word, thereby restoring the original status of IF bit. If the interrupt return re-enables interrupts, and another interrupt is pending, the 80186 will immediately service the highest-priority interrupt pending, i.e., no instructions of the main line program will be executed.

Non-Maskable Interrupt Request (NMI)

A non-maskable interrupt (NMI) is also provided. This interrupt is serviced regardless of the state of the IF bit. A typical use of NMI would be to activate a power failure routine. The activation of this input causes an interrupt with an internally supplied vector value of 2. No external interrupt acknowledge sequence is performed. The IF bit is cleared at the beginning of an NMI interrupt to prevent maskable interrupts from being serviced.

Single-Step Interrupt

The 80186 has an internal interrupt that allows programs to execute one instruction at a time. It is called the single-step interrupt and is controlled by the single-step flag bit (TF) in the Status Word. Once this bit is set, an internal single-step interrupt will occur after the next instruction has been executed. The interrupt clears the TF bit and uses an internally supplied vector of 1. The IRET instruction is used to set the TF bit and transfer control to the next instruction to be single-stepped.

Initialization and Processor Reset

Processor initialization or startup is accomplished by driving the $\overline{RES}$ input pin LOW. $\overline{RES}$ forces the 80186 to terminate all execution and local bus activity. No instruction or bus activity will occur as long as $\overline{RES}$ is active. After $\overline{RES}$ becomes inactive and an internal processing interval elapses, the 80186 begins execution with the instruction at physical location FFFF0(H). $\overline{RES}$ also sets some registers to predefined values as shown in Table 5.

Table 5. 80186 Initial Register State after RESET

Status Word	F002(H)
Instruction Pointer	0000(H)
Code Segment	FFFF(H)
Data Segment	0000(H)
Extra Segment	0000(H)
Stack Segment	0000(H)
Relocation Register	20FF(H)
UMCS	FFFB(H)

80186 CLOCK GENERATOR

The 80186 provides an on-chip clock generator for both internal and external clock generation. The clock generator features a crystal oscillator, a divide-by-two counter, synchronous and asynchronous ready inputs, and reset circuitry.

Oscillator

The oscillator circuit of the 80186 is designed to be used with a parallel resonant fundamental mode crystal. This is used as the time base for the 80186. The crystal frequency selected will be double the CPU clock frequency. Use of an LC or RC circuit is not recommended with this oscillator. If an external oscillator is used, it can be connected directly to input pin X1 in lieu of a crystal. The output of the oscillator is not directly available outside the 80186. The recommended crystal configuration is shown in Figure 8.

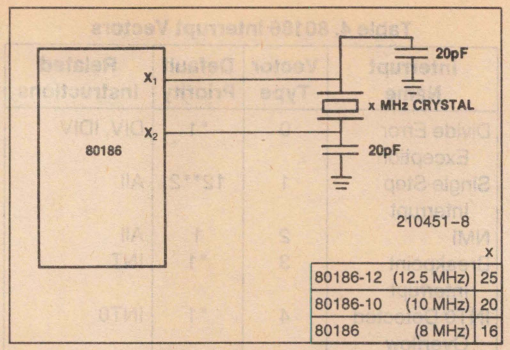


Figure 8. Recommended 80186 Crystal Configuration

The following parameters may be used for choosing a crystal:

Temperature Range:	0 to 70°C
ESR (Equivalent Series Resistance):	30Ω max
C_0 (Shunt Capacitance of Crystal):	7.0 pf max
C_1 (Load Capacitance):	20 pf ± 2 pf
Drive Level:	1 mw max

Clock Generator

The 80186 clock generator provides the 50% duty cycle processor clock for the 80186. It does this by dividing the oscillator output by 2 forming the symmetrical clock. If an external oscillator is used, the state of the clock generator will change on the falling edge of the oscillator signal. The CLKOUT pin provides the processor clock signal for use outside the 80186. This may be used to drive other system components. All timings are referenced to the output clock.

READY Synchronization

The 80186 provides both synchronous and asynchronous ready inputs. Asynchronous ready synchronization is accomplished by circuitry which samples ARDY in the middle of T_2, T_3 and again in the middle of each T_W until ARDY is sampled HIGH. One-half CLKOUT cycle of resolution time is used. Full synchronization is performed only on the rising edge of ARDY, i.e., the falling edge of ARDY must be synchronized to the CLKOUT signal if it will occur during T_2, T_3, or T_W. High-to-LOW transitions of ARDY must be performed synchronously to the CPU clock.

A second ready input (SRDY) is provided to interface with externally synchronized ready signals. This input is sampled at the end of T_2, T_3 and again at the end of each T_W until it is sampled HIGH. By using this input rather than the asynchronous ready input, the half-clock cycle resolution time penalty is eliminated.

This input must satisfy set-up and hold times to guarantee proper operation of the circuit.

In addition, the 80186, as part of the integrated chip-select logic, has the capability to program WAIT states for memory and peripheral blocks. This is discussed in the Chip Select/Ready Logic description.

RESET Logic

The 80186 provides both a $\overline{RES}$ input pin and a synchronized RESET pin for use with other system components. The $\overline{RES}$ input pin on the 80186 is provided with hysteresis in order to facilitate power-on Reset generation via an RC network. RESET is guaranteed to remain active for at least five clocks given a $\overline{RES}$ input of at least six clocks. RESET may be delayed up to two and one-half clocks behind $\overline{RES}$.

Multiple 80186 processors may be synchronized through the $\overline{RES}$ input pin, since this input resets both the processor and divide-by-two internal counter in the clock generator. In order to insure that the divide-by-two counters all begin counting at the same time, the active going edge of $\overline{RES}$ must satisfy a 25 ns setup time before the falling edge of the 80186 clock input. In addition, in order to insure that all CPUs begin executing in the same clock cycle, the reset must satisfy a 25 ns setup time before the rising edge of the CLKOUT signal of all the processors.

LOCAL BUS CONTROLLER

The 80186 provides a local bus controller to generate the local bus control signals. In addition, it employs a HOLD/HLDA protocol for relinquishing the local bus to other bus masters. It also provides control lines that can be used to enable external buffers and to direct the flow of data on and off the local bus.

Memory/Peripheral Control

The 80186 provides ALE, $\overline{RD}$, and $\overline{WR}$ bus control signals. The $\overline{RD}$ and $\overline{WR}$ signals are used to strobe data from memory to the 80186 or to strobe data from the 80186 to memory. The ALE line provides a strobe to address latches for the multiplexed address/data bus. The 80186 local bus controller does not provide a memory/$\overline{I/O}$ signal. If this is required, the user will have to use the $\overline{S2}$ signal (which will require external latching), make the memory and I/O spaces nonoverlapping, or use only the integrated chip-select circuitry.

Transceiver Control

The 80186 generates two control signals to be connected to 8286/8287 transceiver chips. This capability allows the addition of transceivers for extra buffering without adding external logic. These control lines, DT/$\overline{R}$ and $\overline{DEN}$, are generated to control the flow of data through the transceivers. The operation of these signals is shown in Table 6.

Table 6. Transceiver Control Signals Description

Pin Name	Function
$\overline{DEN}$ (Data Enable)	Enables the output drivers of the transceivers. It is active LOW during memory, I/O, or INTA cycles.
DT/$\overline{R}$ (Data Transmit/ Receive)	Determines the direction of travel through the transceivers. A HIGH level directs data away from the processor during write operations, while a LOW level directs data toward the processor during a read operation.

Local Bus Arbitration

The 80186 uses a HOLD/HLDA system of local bus exchange. This provides an asynchronous bus exchange mechanism. This means multiple masters utilizing the same bus can operate at separate clock frequencies. The 80186 provides a single HOLD/HLDA pair through which all other bus masters may gain control of the local bus. This requires external circuitry to arbitrate which external device will gain control of the bus from the 80186 when there is more than one alternate local bus master. When the 80186 relinquishes control of the local bus, it floats $\overline{DEN}$, $\overline{RD}$, $\overline{WR}$, $\overline{S0}$–$\overline{S2}$, $\overline{LOCK}$, AD0–AD15, A16–A19, $\overline{BHE}$, and DT/$\overline{R}$ to allow another master to drive these lines directly.

The 80186 HOLD latency time, i.e., the time between HOLD request and HOLD acknowledge, is a function of the activity occurring in the processor when the HOLD request is received. A HOLD request is the highest-priority activity request which the processor may receive: higher than instruction fetching or internal DMA cycles. However, if a DMA cycle is in progress, the 80186 will complete the transfer before relinquishing the bus. This implies that if a HOLD request is received just as a DMA transfer begins, the HOLD latency time can be as great as 4 bus cycles. This will occur if a DMA word transfer operation is taking place from an odd ad-

dress to an odd address. This is a total of 16 clocks or more, if WAIT states are required. In addition, if locked transfers are performed, the HOLD latency time will be increased by the length of the locked transfer.

Local Bus Controller and Reset

Upon receipt of a RESET pulse from the $\overline{RES}$ input, the local bus controller will perform the following action:

- Drive $\overline{DEN}$, $\overline{RD}$, and $\overline{WR}$ HIGH for one clock cycle, then float.

NOTE:
$\overline{RD}$ is also provided with an internal pull-up device to prevent the processor from inadvertently entering Queue Status mode during reset.

- Drive $\overline{S0}$–$\overline{S2}$ to the passive state (all HIGH) and then float.
- Drive $\overline{LOCK}$ HIGH and then float.
- TRISTATE AD0–15, A16–19, $\overline{BHE}$, DT/$\overline{R}$.
- Drive ALE LOW (ALE is never floated).
- Drive HLDA LOW.

INTERNAL PERIPHERAL INTERFACE

All the 80186 integrated peripherals are controlled via 16-bit registers contained within an internal 256-byte control block. This control block may be mapped into either memory or I/O space. Internal logic will recognize the address and respond to the bus cycle. During bus cycles to internal registers, the bus controller will signal the operation externally (i.e., the $\overline{RD}$, $\overline{WR}$, status, address, data, etc., lines will be driven as in a normal bus cycle), but D_{15-0}, SRDY, and ARDY will be ignored. The base address of the control block must be on an even 256-byte boundary (i.e., the lower 8 bits of the base address are all zeros). All of the defined registers within this control block may be read or written by the 80186 CPU at any time. The location of any register contained within the 256-byte control block is determined by the current base address of the control block.

The control block base address is programmed via a 16-bit relocation register contained within the control block at offset FEH from the base address of the control block (see Figure 9). It provides the upper 12 bits of the base address of the control block. The control block is effectively an internal chip select range and must abide by all the rules concerning chip selects (the chip select circuitry is discussed later in this data sheet). Any access to the 256 bytes of the control block activates an internal chip select. Other chip selects may overlap the control block only if they are programmed to zero wait states and ignore external ready. In addition, bit 12 of this register determines whether the control block will be mapped into I/O or memory space. If this bit is 1, the control block will be located in memory space, whereas if the bit is 0, the control block will be located in I/O space. If the control register block is mapped into I/O space, the upper 4 bits of the base address must be programmed as 0 (since I/O addresses are only 16 bits wide).

In addition to providing relocation information for the control block, the relocation register contains bits which place the interrupt controller into iRMX mode, and cause the CPU to interrupt upon encountering ESC instructions. At RESET, the relocation register is set to 20FFH. This causes the control block to start at FF00H in I/O space. An offset map of the 256-byte control register block is shown in Figure 10.

The integrated 80186 peripherals operate semi-autonomously from the CPU. Access to them for the most part is via software read/write of the control block. Most of these registers can be both read and written. A few dedicated lines, such as interrupts and DMA request provide real-time communication between the CPU and peripherals as in a more conventional system utilizing discrete peripheral blocks. The overall interaction and function of the peripheral blocks has not substantially changed.

CHIP-SELECT/READY GENERATION LOGIC

The 80186 contains logic which provides programmable chip-select generation for both memories and peripherals. In addition, it can be programmed to provide READY (or WAIT state) generation. It can also povide latched address bits A1 and A2. The chip-select lines are active for all memory and I/O cycles in their programmed areas, whether they be generated by the CPU or by the integrated DMA unit.

Memory Chip Selects

The 80186 provides 6 memory chip select outputs for 3 address areas; upper memory, lower memory, and midrange memory. One each is provided for upper memory and lower memory, while four are provided for midrange memory.

The range for each chip select is user-programmable and can be set to 2K, 4K, 8K, 16K, 32K, 64K, 128K (plus 1K and 256K for upper and lower chip selects). In addition, the beginning or base address

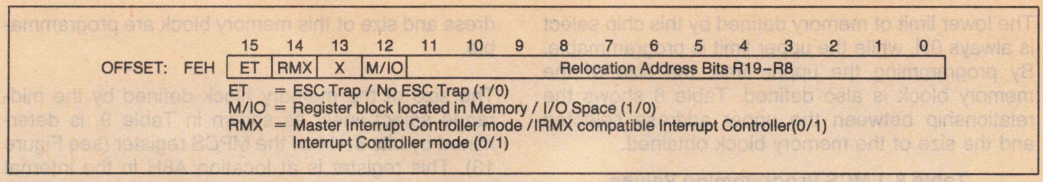


Figure 9. Relocation Register

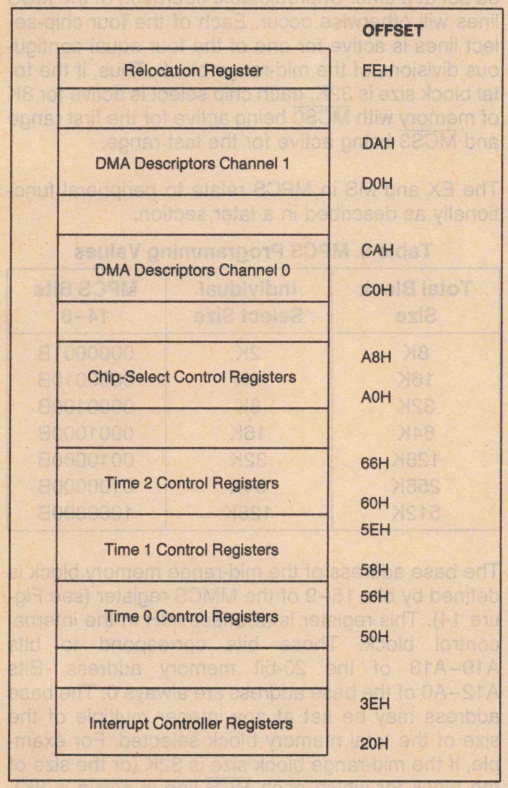


Figure 10. Internal Register Map

of the midrange memory chip select may also be selected. Only one chip select may be programmed to be active for any memory location at a time. All chip select sizes are in bytes, whereas 80186 memory is arranged in words. This means that if, for example, 16 64K x 1 memories are used, the memory block size will be 128K, not 64K.

Upper Memory $\overline{CS}$

The 80186 provides a chip select, called $\overline{UCS}$, for the top of memory. The top of memory is usually used as the system memory because after reset the 80186 begins executing at memory location FFFF0H.

The upper limit of memory defined by this chip select is always FFFFFH, while the lower limit is programmable. By programming the lower limit, the size of the select block is also defined. Table 7 shows the relationship between the base address selected and the size of the memory block obtained.

Table 7. UMCS Programming Values

Starting Address (Base Address)	Memory Block Size	UMCS Value (Assuming R0=R1=R2=0)
FFC00	1K	FFF8H
FF800	2K	FFB8H
FF000	4K	FF38H
FE000	8K	FE38H
FC000	16K	FC38H
F8000	32K	F838H
F0000	64K	F038H
E0000	128K	E038H
C0000	256K	C038H

The lower limit of this memory block is defined in the UMCS register (see Figure 11). This register is at offset A0H in the internal control block. The legal values for bits 6–13 and the resulting starting address and memory block sizes are given in Table 7. Any combination of bits 6–13 not shown in Table 7 will result in undefined operation. After reset, the UMCS register is programmed for a 1K area. It must be reprogrammed if a larger upper memory area is desired.

Any internally generated 20-bit address whose upper 16 bits are greater than or equal to UMCS (with bits 0–5 "0") will cause UCS to be activated. UMCS bits R2–R0 are used to specify READY mode for the area of memory defined by this chip-select register, as explained below.

Lower Memory $\overline{CS}$

The 80186 provides a chip select for low memory called $\overline{LCS}$. The bottom of memory contains the interrupt vector table, starting at location 00000H.

The lower limit of memory defined by this chip select is always 0H, while the upper limit is programmable. By programming the upper limit, the size of the memory block is also defined. Table 8 shows the relationship between the upper address selected and the size of the memory block obtained.

Table 8. LMCS Programming Values

Upper Address	Memory Block Size	LMCS Value (Assuming R0=R1=R2=0)
003FFH	1K	0038H
007FFH	2K	0078H
00FFFH	4K	00F8H
01FFFH	8K	01F8H
03FFFH	16K	03F8H
07FFFH	32K	07F8H
0FFFFH	64K	0FF8H
1FFFFH	128K	1FF8H
3FFFFH	256K	3FF8H

The upper limit of this memory block is defined in the LMCS register (see Figure 12). This register is at offset A2H in the internal control block. The legal values for bits 6–15 and the resulting upper address and memory block sizes are given in Table 8. Any combination of bits 6–15 not shown in Table 8 will result in undefined operation. After reset, the LMCS register value is undefined. However, the $\overline{LCS}$ chip-select line will not become active until the LMCS register is accessed.

Any internally generated 20-bit address whose upper 16 bits are less than or equal to LMCS (with bits 0–5 "1") will cause $\overline{LCS}$ to be active. LMCS register bits R2–R0 are used to specify the READY mode for the area of memory defined by this chip-select register.

Mid-Range Memory $\overline{CS}$

The 80186 provides four $\overline{MCS}$ lines which are active within a user-locatable memory block. This block can be located anywhere within the 80186 1M byte memory address space exclusive of the areas defined by $\overline{UCS}$ and $\overline{LCS}$. Both the base address and size of this memory block are programmable.

The size of the memory block defined by the mid-range select lines, as shown in Table 9, is determined by bits 8–14 of the MPCS register (see Figure 13). This register is at location A8H in the internal control block. One and only one of bits 8–14 must be set at a time. Unpredictable operation of the $\overline{MCS}$ lines will otherwise occur. Each of the four chip-select lines is active for one of the four equal contiguous divisions of the mid-range block. Thus, if the total block size is 32K, each chip select is active for 8K of memory with $\overline{MCS0}$ being active for the first range and $\overline{MCS3}$ being active for the last range.

The EX and MS in MPCS relate to peripheral functionally as described in a later section.

Table 9. MPCS Programming Values

Total Block Size	Individual Select Size	MPCS Bits 14–8
8K	2K	0000001B
16K	4K	0000010B
32K	8K	0000100B
64K	16K	0001000B
128K	32K	0010000B
256K	64K	0100000B
512K	128K	1000000B

The base address of the mid-range memory block is defined by bits 15–9 of the MMCS register (see Figure 14). This register is at offset A6H in the internal control block. These bits correspond to bits A19–A13 of the 20-bit memory address. Bits A12–A0 of the base address are always 0. The base address may be set at any integer multiple of the size of the total memory block selected. For example, if the mid-range block size is 32K (or the size of the block for which each $\overline{MCS}$ line is active is 8K), the block could be located at 10000H or 18000H, but not at 14000H, since the first few integer multiples of a 32K memory block are 0H, 8000H, 10000H, 18000H, etc. After reset, the contents of both of these registers is undefined. However, none of the $\overline{MCS}$ lines will be active until both the MMCS and MPCS registers are accessed.

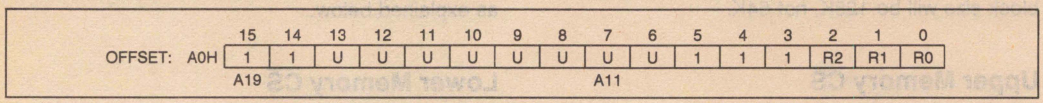


Figure 11. UMCS Register

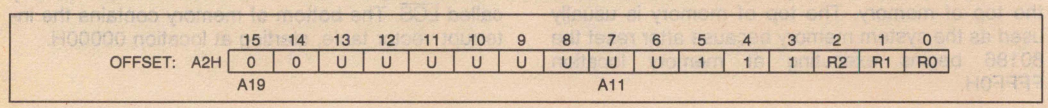


Figure 12. LMCS Register

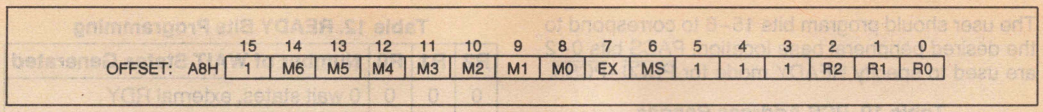


Figure 13. MPCS Register

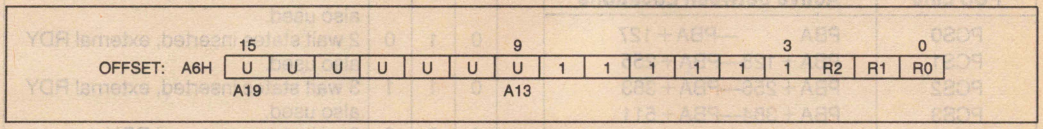


Figure 14. MMCS Register

MMCS bits R2–R0 specify READY mode of operation for all mid-range chip selects. All devices in mid-range memory must use the same number of WAIT states.

The 512K block size for the mid-range memory chip selects is a special case. When using 512K, the base address would have to be at either locations 00000H or 80000H. If it were to be programmed at 00000H when the $\overline{\text{LCS}}$ line was programmed, there would be an internal conflict between the $\overline{\text{LCS}}$ ready generation logic and the $\overline{\text{MCS}}$ ready generation logic. Likewise, if the base address were programmed at 80000H, there would be a conflict with the $\overline{\text{UCS}}$ ready generation logic. Since the $\overline{\text{LCS}}$ chip-select line does not become active until programmed, while the $\overline{\text{UCS}}$ line is active at reset, the memory base can be set only at 00000H. If this base address is selected, however, the $\overline{\text{LCS}}$ range must not be programmed.

Peripheral Chip Selects

The 80186 can generate chip selects for up to seven peripheral devices. These chip selects are active for seven contiguous blocks of 128 bytes above a programmable base address. This base address may be located in either memory or I/O space.

Seven $\overline{\text{CS}}$ lines called $\overline{\text{PCS0}}$–6 are generated by the 80186. The base address is user-programmable; however it can only be a multiple of 1K bytes, i.e., the least significant 10 bits of the starting address are always 0.

$\overline{\text{PCS5}}$ and $\overline{\text{PCS6}}$ can also be programmed to provide latched address bits A1, A2. If so programmed, they cannot be used as peripheral selects. These outputs can be connected directly to the A0, A1 pins used for selecting internal registers of 8-bit peripheral chips. This scheme simplifies the hardware interface because the 8-bit registers of peripherals are simply treated as 16-bit registers located on even boundaries in I/O space or memory space where only the lower 8-bits of the register are significant: the upper 8-bits are "don't cares."

The starting address of the peripheral chip-select block is defined by the PACS register (see Figure 15). This register is located at offset A4H in the internal control block. Bits 15–6 of this register correspond to bits 19–10 of the 20-bit Programmable Base Address (PBA) of the peripheral chip-select block. Bits 9–0 of the PBA of the peripheral chip-select block are all zeros. If the chip-select block is located in I/O space, bits 12–15 must be programmed zero, since the I/O address is only 16 bits wide. Table 10 shows the address range of each peripheral chip select with respect to the PBA contained in PACS register.

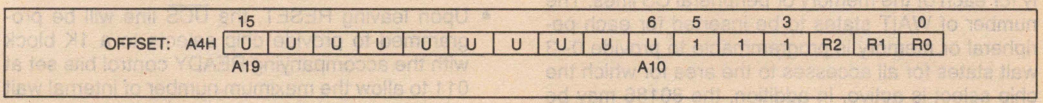


Figure 15. PACS Register

The user should program bits 15–6 to correspond to the desired peripheral base location. PACS bits 0–2 are used to specify READY mode for $\overline{PSC0}$–$\overline{PCS3}$.

Table 10. PCS Address Ranges

PCS Line	Active between Locations
PCS0	PBA —PBA + 127
PCS1	PBA + 128—PBA + 255
PCS2	PBA + 256—PBA + 383
PCS3	PBA + 384—PBA + 511
PCS4	PBA + 512—PBA + 639
PCS5	PBA + 640—PBA + 767
PCS6	PBA + 768—PBA + 895

The mode of operation of the peripheral chip selects is defined by the MPCS register (which is also used to set the size of the mid-range memory chip-select block, see Figure 16). This register is located at offset A8H in the internal control block. Bit 7 is used to select the function of $\overline{PCS5}$ and $\overline{PCS6}$, while bit 6 is used to select whether the peripheral chip selects are mapped into memory or I/O space. Table 11 describes the programming of these bits. After reset, the contents of both the MPCS and the PACS registers are undefined, however none of the PCS lines will be active until both of the MPCS and PACS registers are accessed.

Table 11. MS, EX Programming Values

Bit	Description
MS	1 = Peripherals mapped into memory space.
	0 = Peripherals mapped into I/O space.
EX	0 = 5 $\overline{PCS}$ lines. A1, A2 provided.
	1 = 7 $\overline{PCS}$ lines. A1, A2 are not provided.

MPCS bits 0–2 are used to specify READY mode for $\overline{PCS4}$–$\overline{PCS6}$ as outlined below.

READY Generation Logic

The 80186 can generate a "READY" signal internally for each of the memory or peripheral $\overline{CS}$ lines. The number of WAIT states to be inserted for each peripheral or memory is programmable to provide 0–3 wait states for all accesses to the area for which the chip select is active. In addition, the 80186 may be programmed to either ignore external READY for each chip-select range individually or to factor external READY with the integrated ready generator.

READY control consists of 3 bits for each $\overline{CS}$ line or group of lines generated by the 80186. The interpretation of the ready bits is shown in Table 12.

Table 12. READY Bits Programming

R2	R1	R0	Number of WAIT States Generated
0	0	0	0 wait states, external RDY also used.
0	0	1	1 wait state inserted, external RDY also used.
0	1	0	2 wait states inserted, external RDY also used.
0	1	1	3 wait states inserted, external RDY also used.
1	0	0	0 wait states, external RDY ignored.
1	0	1	1 wait state inserted, external RDY ignored.
1	1	0	2 wait states inserted, external RDY ignored.
1	1	1	3 wait states inserted, external RDY ignored.

The internal ready generator operates in parallel with external READY, not in series if the external READY is used (R2 = 0). This means, for example, if the internal generator is set to insert two wait states, but activity on the external READY lines will insert four wait states, the processor will only insert four wait states, not six. This is because the two wait states generated by the internal generator overlapped the first two wait states generated by the external ready signal. Note that the external ARDY and SRDY lines are always ignored during cycles accessing internal peripherals.

R2–R0 of each control word specifies the READY mode for the corresponding block, with the exception of the peripheral chip selects: R2–R0 of PACS set the $\overline{PCS0}$–3 READY mode, R2–R0 of MPCS set the $\overline{PCS4}$–6 READY mode.

Chip Select/Ready Logic and Reset

Upon reset, the Chip-Select/Ready Logic will perform the following actions:

- All chip-select outputs will be driven HIGH.
- Upon leaving RESET, the $\overline{UCS}$ line will be programmed to provide chip selects to a 1K block with the accompanying READY control bits set at 011 to allow the maximum number of internal wait states in conjunction with external Ready consideration (i.e., UMCS resets to FFFBH).
- No other chip select or READY control registers have any predefined values after RESET. They will not become active until the CPU accesses their control registers. Both the PACS and MPCS registers must be accessed before the $\overline{PCS}$ lines will become active.

OFFSET: A8H	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	M6	M5	M4	M3	M2	M1	M0	EX	MS	1	1	1	R2	R1	R0

Figure 16. MPCS Register

DMA CHANNELS

The 80186 DMA controller provides two independent high-speed DMA channels. Data transfers can occur between memory and I/O spaces (e.g., Memory to I/O) or within the same space (e.g., Memory to Memory or I/O to I/O). Data can be transferred either in bytes (8 bits) or in words (16 bits) to or from even or odd addresses. Each DMA channel maintains both a 20-bit source and destination pointer which can be optionally incremented or decremented after each data transfer (by one or two depending on byte or word transfers). Each data transfer consumes 2 bus cycles (a minimum of 8 clocks), one cycle to fetch data and the other to store data. This provides a maximum data transfer rate of one Mword/sec or 2 MBytes/sec.

DMA Operation

Each channel has six registers in the control block which define each channel's specific operation. The control registers consist of a 20-bit Source pointer (2 words), a 20-bit destination pointer (2 words), a 16-bit Transfer Counter, and a 16-bit Control Word. The format of the DMA Control Blocks is shown in Table 13. The Transfer Count Register (TC) specifies the number of DMA transfers to be performed. Up to 64K byte or word transfers can be performed with automatic termination. The Control Word defines the channel's operation (see Figure 18). All registers may be modified or altered during any DMA activity. Any changes made to these registers will be reflected immediately in DMA operation.

Table 13. DMA Control Block Format

Register Name	Register Address	
	Ch. 0	Ch. 1
Control Word	CAH	DAH
Transfer Count	C8H	D8H
Destination Pointer (upper 4 bits)	C6H	D6H
Destination Pointer	C4H	D4H
Source Pointer (upper 4 bits)	C2H	D2H
Source Pointer	C0H	D0H

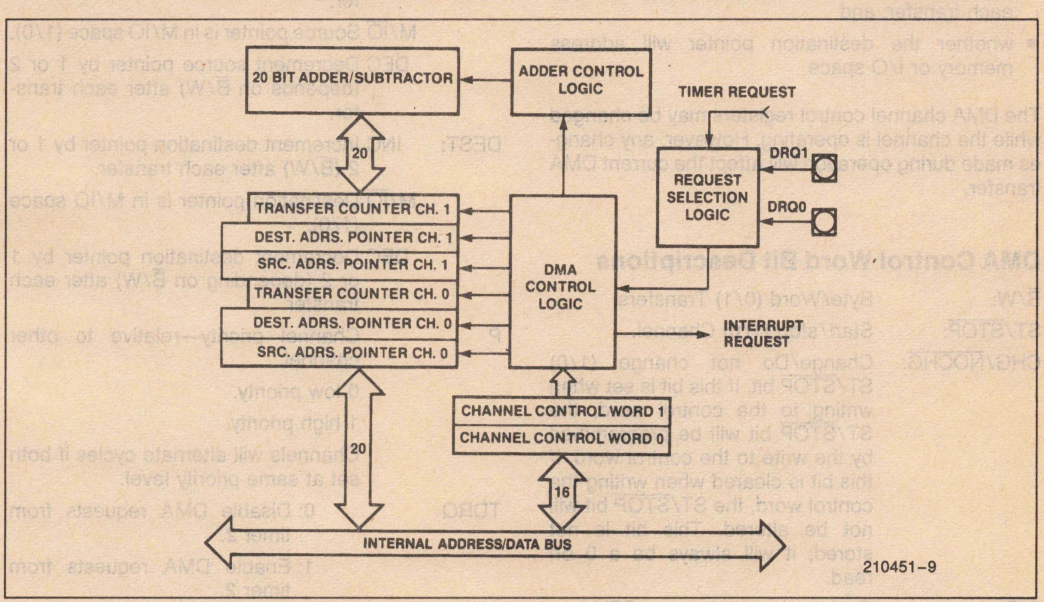


Figure 17. DMA Unit Block Diagram

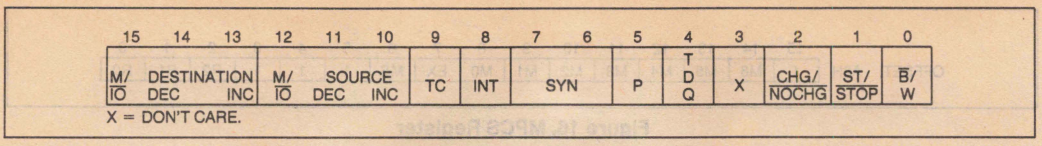


Figure 18. DMA Control Register

DMA Channel Control Word Register

Each DMA Channel Control Word determines the mode of operation for the particular 81086 DMA channel. This register specifies:

- the mode of synchronization;
- whether bytes or words will be transferred;
- whether interrupts will be generated after the last transfer;
- whether DMA activity will cease after a programmed number of DMA cycles;
- the relative priority of the DMA channel with respect to the other DMA channel;
- whether the source pointer will be incremented, decremented, or maintained constant after each transfer;
- whether the source pointer addresses memory or I/O space;
- whether the destination pointer will be incremented, decremented, or maintained constant after each transfer; and
- whether the destination pointer will address memory or I/O space.

The DMA channel control registers may be changed while the channel is operating. However, any changes made during operation will affect the current DMA transfer.

DMA Control Word Bit Descriptions

$\overline{B}$/W: Byte/Word (0/1) Transfers.

ST/$\overline{STOP}$: Start/stop (1/0) Channel.

CHG/$\overline{NOCHG}$: Change/Do not change (1/0) ST/$\overline{STOP}$ bit. If this bit is set when writing to the control word, the ST/$\overline{STOP}$ bit will be programmed by the write to the control word. If this bit is cleared when writing the control word, the ST/$\overline{STOP}$ bit will not be altered. This bit is not stored; it will always be a 0 on read.

INT: Enable Interrupts to CPU on Transfer Count termination.

TC: If set, DMA will terminate when the contents of the Transfer Count register reach zero. The ST/$\overline{STOP}$ bit will also be reset at this point if TC is set. If this bit is cleared, the DMA unit will decrement the transfer count register for each DMA cycle, but the DMA transfer will not stop when the contents of the TC register reach zero.

NOTE:

The ST bit will be cleared automatically when the contents of the TC register reach zero regardless of the state of the TC bit.

SYN (2 bits)
00 No synchronization.
01 Source synchronization.
10 Destination synchronization.
11 Unused.

SOURCE:INC Increment source pointer by 1 or 2 (depends on $\overline{B}$/W) after each transfer.

M/$\overline{IO}$ Source pointer is in M/IO space (1/0).

DEC Decrement source pointer by 1 or 2 (depends on $\overline{B}$/W) after each transfer.

DEST: INC Increment destination pointer by 1 or 2 ($\overline{B}$/W) after each transfer.

M/$\overline{IO}$ Destination pointer is in M/IO space (1/0).

DEC Decrement destination pointer by 1 or 2 (depending on $\overline{B}$/W) after each transfer.

P Channel priority—relative to other channel.
0 low priority.
1 high priority.
Channels will alternate cycles if both set at same priority level.

TDRQ
0: Disable DMA requests from timer 2.
1: Enable DMA requests from timer 2.

Bit 3 Bit 3 is not used.

If both INC and DEC are specified for the same pointer, the pointer will remain constant after each cycle.

DMA Destination and Source Pointer Registers

Each DMA channel maintains a 20-bit source and a 20-bit destination pointer. Each of these pointers takes up two full 16-bit registers in the peripheral control block. The lower four bits of the upper register contain the upper four bits of the 20-bit physical address (see Figure 18a). These pointers may be individually incremented or decremented after each transfer. If word transfers are performed the pointer is incremented or decremented by two. Each pointer may point into either memory or I/O space. Since the DMA channels can perform transfers to or from odd addresses, there is no restriction on values for the pointer registers. Higher transfer rates can be obtained if all word transfers are performed to even addresses, since this will allow data to be accessed in a single memory access.

DMA Transfer Count Register

Each DMA channel maintains a 16-bit transfer count register (TC). This register is decremented after every DMA cycle, regardless of the state of the TC bit in the DMA Control Register. If the TC bit in the DMA control word is set or unsynchronized transfers are programmed, however, DMA activity will terminate when the transfer count register reaches zero.

DMA Requests

Data transfers may be either source or destination synchronized, that is either the source of the data or the destination of the data may request the data transfer. In addition, DMA transfers may be unsynchronized; that is, the transfer will take place continually until the correct number of transfers has occurred. When source or unsynchronized transfers are performed, the DMA channel may begin another transfer immediately after the end of a previous DMA transfer. This allows a complete transfer to take place every 2 bus cycles or eight clock cycles (assuming no wait states). No prefetching occurs when destination synchronization is performed, however. Data will not be fetched from the source address until the destination device signals that it is ready to receive it. When destination synchronized transfers are requested, the DMA controller will relinquish control of the bus after every transfer. If no other bus activity is initiated, another DMA cycle will begin after two processor clocks. This is done to allow the destination device time to remove its request if another transfer is not desired. Since the DMA controller will relinquish the bus, the CPU can initiate a bus cycle. As a result, a complete bus cycle will often be inserted between destination synchronized transfers. These lead to the maximum DMA transfer rates shown in Table 14.

Table 14. Maximum DMA Transfer Rates

Type of Synchronization Selected	CPU Running	CPU Halted
Unsynchronized	2MBytes/sec	2MBytes/sec
Source Synch	2MBytes/sec	2MBytes/sec
Destination Synch	1.3MBytes/sec	1.5MByte/sec

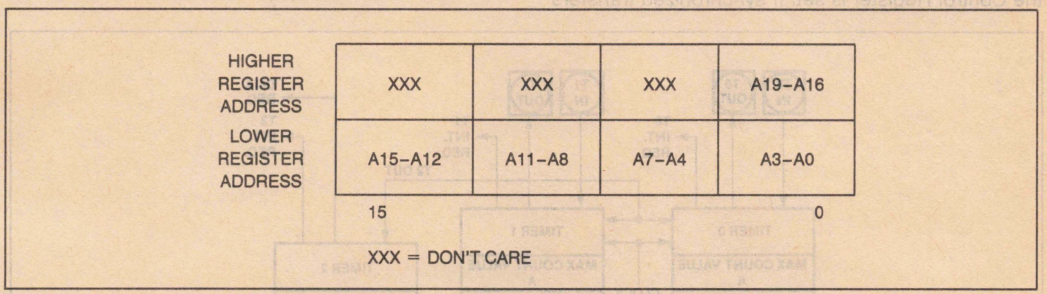


Figure 18a. DMA Memory Pointer Register Format

DMA Acknowledge

No explicit DMA acknowledge pulse is provided. Since both source and destination pointers are maintained, a read from a requesting source, or a write to a requesting destination, should be used as the DMA acknowledge signal. Since the chip-select lines can be programmed to be active for a given block of memory or I/O space, and the DMA pointers can be programmed to point to the same given block, a chip-select line could be used to indicate a DMA acknowledge.

DMA Priority

The DMA channels may be programmed such that one channel is always given priority over the other, or they may be programmed such as to alternate cycles when both have DMA requests pending. DMA cycles always have priority over internal CPU cycles except between locked memory accesses or word accesses the odd memory locations; however, an external bus hold takes priority over an internal DMA cycle. Because an interrupt request cannot suspend a DMA operation and the CPU cannot access memory during a DMA cycle, interrupt latency time will suffer during sequences of continuous DMA cycles. An NMI request, however, will cause all internal DMA activity to halt. This allows the CPU to quickly respond to the NMI request.

DMA Programming

DMA cycles will occur whenever the ST/STOP bit of the Control Register is set. If synchronized transfers are programmed, a DRQ must also have been generated. Therefore the source and destination transfer pointers, and the transfer count register (if used) must be programmed before this bit is set.

Each DMA register may be modified while the channel is operating. If the CHG/NOCHG bit is cleared when the control register is written, the ST/STOP bit of the control register will not be modified by the write. If multiple channel registers are modified, it is recommended that a LOCKED string transfer be used to prevent a DMA transfer from occurring between updates to the channel registers.

DMA Channels and Reset

Upon RESET, the DMA channels will perform the following actions:

- The Start/Stop bit for each channel will be reset to STOP.
- Any transfer in progress is aborted.

TIMERS

The 80186 provides three internal 16-bit programmable timers (see Figure 19). Two of these are highly flexible and are connected to four external pins (2 per timer). They can be used to count external events, time external events, generate nonrepetitive waveforms, etc. The third timer is not connected to any external pins, and is useful for real-time coding and time delay applications. In addition, this third timer can be used as a prescaler to the other two, or as a DMA request source.

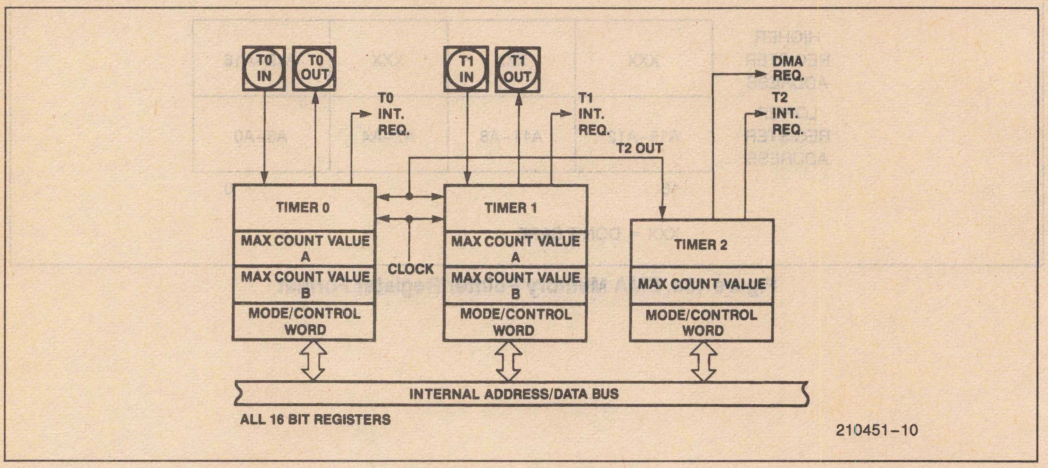


Figure 19. Timer Block Diagram

Timer Operation

The timers are controlled by 11 16-bit registers in the internal peripheral control block. The configuration of these registers is shown in Table 15. The count register contains the current value of the timer. It can be read or written at any time independent of whether the timer is running or not. The value of this register will be incremented for each timer event. Each of the timers is equipped with a MAX COUNT register, which defines the maximum count the timer will reach. After reaching the MAX COUNT register value, the timer count value will reset to zero during that same clock, i.e., the maximum count value is never stored in the count register itself. Timers 0 and 1 are, in addition, equipped with a second MAX COUNT register, which enables the timers to alternate their count between two different MAX COUNT values programmed by the user. If a single MAX COUNT register is used, the timer output pin will switch LOW for a single clock, 1 clock after the maximum count value has been reached. In the dual MAX COUNT register mode, the output pin will indicate which MAX COUNT register is currently in use, thus allowing nearly complete freedom in selecting waveform duty cycles. For the timers with two MAX COUNT registers, the RIU bit in the control register determines which is used for the comparison.

Each timer gets serviced every fourth CPU-clock cycle, and thus can operate at speeds up to one-quarter the internal clock frequency (one-eighth the crystal rate). External clocking of the timers may be done at up to a rate of one-quarter of the internal CPU-clock rate (2 MHz for an 8 MHz CPU clock). Due to internal synchronization and pipelining of the timer circuitry, a timer output may take up to 6 clocks to respond to any individual clock or gate input.

Since the count registers and the maximum count registers are all 16 bits wide, 16 bits of resolution are provided. Any Read or Write access to the timers will add one wait state to the minimum four-clock bus cycle, however. This is needed to synchronize and coordinate the internal data flows between the internal timers and the internal bus.

The timers have several programmable options.

- All three timers can be set to halt or continue on a terminal count.
- Timers 0 and 1 can select between internal and external clocks, alternate between MAX COUNT registers and be set to retrigger on external events.
- The timers may be programmed to cause an interrupt on terminal count.

These options are selectable via the timer mode/control word.

Timer Mode/Control Register

The mode/control register (see Figure 20) allows the user to program the specific mode of operation or check the current programmed status for any of the three integrated timers.

Table 15. Timer Control Block Format

Register Name	Register Offset		
	Tmr. 0	Tmr. 1	Tmr. 2
Mode/Control Word	56H	5EH	66H
Max Count B	54H	5CH	not present
Max Count A	52H	5AH	62H
Count Register	50H	58H	60H

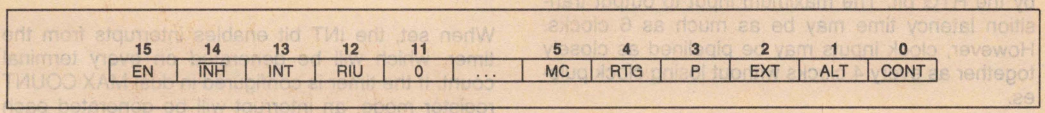


Figure 20. Timer Mode/Control Register

ALT:

The ALT bit determines which of two MAX COUNT registers is used for count comparison. If ALT = 0, register A for that timer is always used, while if ALT = 1, the comparison will alternate between register A and register B when each maximum count is reached. This alternation allows the user to change one MAX COUNT register while the other is being used, and thus provides a method of generating non-repetitive waveforms. Square waves and pulse outputs of any duty cycle are a subset of available signals obtained by not changing the final count registers. The ALT bit also determines the function of the timer output pin. If ALT is zero, the output pin will go LOW for one clock, the clock after the maximum count is reached. If ALT is one, the output pin will reflect the current MAX COUNT register being used (0/1 for B/A).

CONT:

Setting the CONT bit causes the associated timer to run continuously, while resetting it causes the timer to halt upon maximum count. If COUNT = 0 and ALT = 1, the timer will count to the MAX COUNT register A value, reset, count to the register B value, reset, and halt.

EXT:

The external bit selects between internal and external clocking for the timer. The external signal may be asynchronous with respect to the 80186 clock. If this bit is set, the timer will count LOW-to-HIGH transitions on the input pin. If cleared, it will count an internal clock while using the input pin for control. In this mode, the function of the external pin is defined by the RTG bit. The maximum input to output transition latency time may be as much as 6 clocks. However, clock inputs may be pipelined as closely together as every 4 clocks without losing clock pulses.

P:

The prescaler bit is ignored unless internal clocking has been selected (EXT = 0). If the P bit is a zero, the timer will count at one-fourth the internal CPU clock rate. If the P bit is a one, the output of timer 2 will be used as a clock for the timer. Note that the user must initialize and start timer 2 to obtain the prescaled clock.

RTG:

Retrigger bit is only active for internal clocking (EXT = 0). In this case it determines the control function provided by the input pin.

If RTG = 0, the input level gates the internal clock on and off. If the input pin is HIGH, the timer will count; if the input pin is LOW, the timer will hold its value. As indicated previously, the input signal may be asynchronous with respect to the 80186 clock.

When RTG = 1, the input pin detects LOW-to-HIGH transitions. The first such transition starts the timer running, clearing the timer value to zero on the first clock, and then incrementing thereafter. Further transitions on the input pin will again reset the timer to zero, from which it will start counting up again. If CONT = 0, when the timer has reached maximum count, the EN bit will be cleared, inhibiting further timer activity.

EN:

The enable bit provides programmer control over the timer's RUN/HALT status. When set, the timer is enabled to increment subject to the input pin constraints in the internal clock mode (discussed previously). When cleared, the timer will be inhibited from counting. All input pin transistions during the time EN is zero will be ignored. If CONT is zero, the EN bit is automatically cleared upon maximum count.

INH:

The inhibit bit allows for selective updating of the enable (EN) bit. If INH is a one during the write to the mode/control word, then the state of the EN bit will be modified by the write. If INH is a zero during the write, the EN bit will be unaffected by the operation.This bit is not stored; it will always be a 0 on a read.

INT:

When set, the INT bit enables interrupts from the timer, which will be generated on every terminal count. If the timer is configured in dual MAX COUNT register mode, an interrupt will be generated each time the value in MAX COUNT register A is reached, and each time the value in MAX COUNT register B is reached. If this enable bit is cleared after the interrupt request has been generated, but before a pending interrupt is serviced, the interrupt request will still be in force. (The request is latched in the Interrupt Controller).

MC:

The Maximum Count bit is set whenever the timer reaches its final maximum count value. If the timer is configured in dual MAX COUNT register mode, this bit will be set each time the value in MAX COUNT register A is reached, and each time the value in MAX COUNT register B is reached. This bit is set

regardless of the timer's interrupt-enable bit. The MC bit gives the user the ability to monitor timer status through software instead of through interrupts.

Programmer intervention is required to clear this bit.

RIU:

The Register In Use bit indicates which MAX COUNT register is currently being used for comparison to the timer count value. A zero value indicates register A. The RIU bit cannot be written, i.e., its value is not affected when the control register is written. It is always cleared when the ALT bit is zero.

Not all mode bits are provided for timer 2. Certain bits are hardwired as indicated below:

ALT = 0, EXT = 0, P = 0, RTG = 0, RIU = 0

Count Registers

Each of the three timers has a 16-bit count register. The current contents of this register may be read or written by the processor at any time. If the register is written into while the timer is counting,the new value will take effect in the current count cycle.

Max Count Registers

Timers 0 and 1 have two MAX COUNT registers, while timer 2 has a single MAX COUNT register. These contain the number of events the timer will count. In timers 0 and 1, the MAX COUNT register used can alternate between the two max count values whenever the current maximum count is reached. The condition which causes a timer to reset is equivalent between the current count value and the max count being used. This means that if the count is changed to be above the max count value, or if the max count value is changed to be below the current value, the timer will not reset to zero, but rather will count to its maximum value, "wrap around" to zero, then count until the max count is reached.

Timers and Reset

Upon RESET, the Timers will perform the following actions:

- All EN (Enable) bits are reset preventing timer counting.
- All SEL (Select) bits are reset to zero. This selects MAX COUNT register A, resulting in the Timer Out pins going HIGH upon RESET.

INTERRUPT CONTROLLER

The 80186 can receive interrupts from a number of sources, both internal and external. The internal interrupt controller serves to merge these requests on a priority basis, for individual service by the CPU.

Internal interrupt sources (Timers and DMA channels) can be disabled by their own control registers or by mask bits within the interrupt controller. The 80186 interrupt controller has its own control register that set the mode of operation for the controller.

The interrupt controller will resolve priority among requests that are pending simultaneously. Nesting is provided so interrupt service routines for lower priority interrupts may themselves be interrupted by higher priority interrupts. A block diagram of the interrupt controller is shown in Figure 21.

The interrupt controller has a special iRMX 86 compatibility mode that allows the use of the 80186 within the iRMX 86 operating system interrupt structure. The conroller is set in this mode by setting bit 14 in the peripheral control block relocation register (see iRMX 86 Compatibility Mode section). In this mode, the internal 80186 interrupt controller functions as a "slave" controller to an external "master" controller. Special initialization software must be included to properly set up the 80186 interrupt controller in iRMX 86 mode.

MASTER MODE OPERATION

Interrupt Controller External Interface

For external interrupt sources, five dedicated pins are provided. One of these pins is dedicated to NMI, non-maskable interrupt. This is typically used for power-fail interrupts, etc. The other four pins may function either as four interrupt input lines with internally generated interrupt vectors, as an interrupt line and an interrupt acknowledge line (called the "cascade mode") along with two other input lines with internally generated interrupt vectors, or as two interrupt input lines and two dedicated interrupt acknowledge output lines. When the interrupt lines are configured in cascade mode, the 80186 interrupt controller will not generate internal interrupt vectors.

External sources in the cascade mode use externally generated interrupt vectors. When an interrupt is acknowledged, two $\overline{\text{INTA}}$ cycles are initiated and the vector is read into the 80186 on the second cycle. The capability to interface to external 8259A programmable interrupt controllers is thus provided when the inputs are configured in cascade mode.

Interrupt Controller Modes of Operation

The basic modes of operation of the interrupt controller in master mode are similar to the 8259A. The interrupt controller responds indentically to internal interrupts in all three modes: the difference is only in the interpretation of function of the four external interrupt pins. The interrupt controller is set into one of these three modes by programming the correct bits in the INT0 and INT1 control registers. The modes of interrupt controller operation are as follows:

Fully Nested Mode

When in the fully nested mode four pins are used as direct interrupt requests. The vectors for these four inputs are generated internally. An in-service bit is provided for every interrupt source. If a lower-priority device requests an interrupt while the in service bit (IS) is set, no interrupt will be generated by the interrupt controller. In addition, if another interrupt request occurs from the same interrupt source while the in-service bit is set, no interrupt will be generated by the interrupt controller. This allows interrupt service routines to operate with interrupts enabled without being themselves interrupted by lower-priority interrupts. Since interrupts are enabled, higher-priority interrupts will be serviced.

When a service routine is completed, the proper IS bit must be reset by writing the proper pattern to the EOI register. This is required to allow subsequent interrupts from this interrupt source and to allow servicing of lower-priority interrupts. An EOI command is issued at the end of the service routine just before the issuance of the return from interrupt instruction. If the fully nested structure has been upheld, the next highest-priority source with its IS bit set is then serviced.

Cascade Mode

The 80186 has four interrupt pins and two of them have dual functions. In the fully nested mode the four pins are used as direct interrupt inputs and the corresponding vectors are generated internally. In the cascade mode, the four pins are configured into interrupt input-dedicated acknowledge signal pairs. The interconnection is shown in Figure 22. INT0 is an interrupt input interfaced to an 8259A, while INT2/$\overline{\text{INTA0}}$ serves as the dedicated interrupt acknowledge signal to that peripheral. The same is true for INT1 and INT3/$\overline{\text{INTA1}}$. Each pair can selectively be placed in the cascade or non-cascade mode by programming the proper value into INT0 and INT1 control registers. The use of the dedicated acknowledge signals eliminates the need for the use of external logic to generate $\overline{\text{INTA}}$ and device select signals.

The primary cascade mode allows the capability to serve up to 128 external interrupt sources through the use of external master and slave 8259As. Three levels of priority are created, requiring priority resolution in the 80186 interrupt controller, the master 8259As, and the slave 8259As. If an external interrupt is serviced, one IS bit is set at each of these levels. When the interrupt service routine is completed, up to three end-of-interrupt commands must be issued by the programmer.

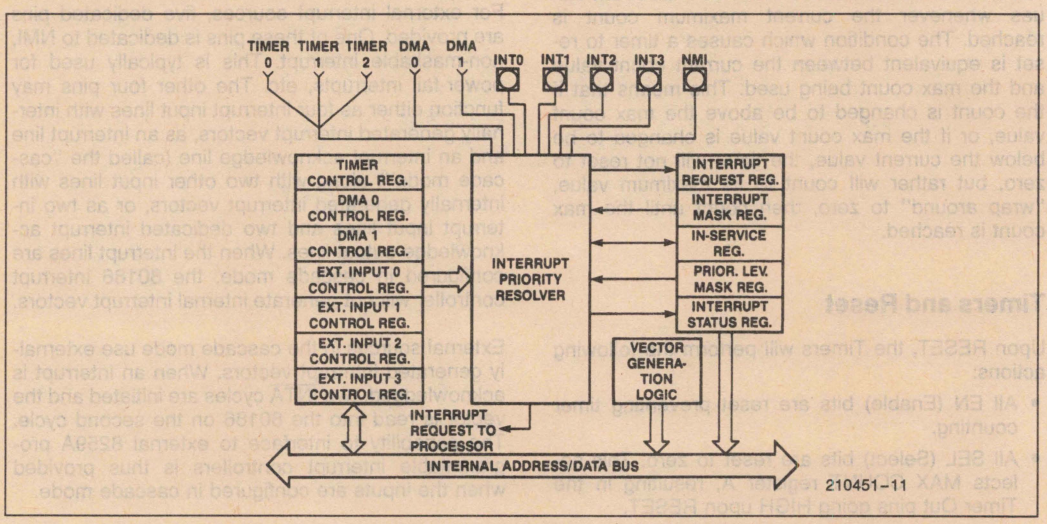


Figure 21. Interrupt Controller Block Diagram

Special Fully Nested Mode

This mode is entered by setting the SFNM bit in INT0 or INT1 control register. It enables complete nestability with external 8259A masters. Normally, an interrupt request from an interrupt source will not be recognized unless the in-service bit for that source is reset. If more than one interrupt source is connected to an external interrupt controller, all of the interrupts will be funneled through the same 80186 interrupt request pin. As a result, if the external interrupt controller receives a higher-priority interrupt, its interrupt will not be recognized by the 80186 controller until the 80186 in-service bit is reset. In special fully nested mode, the 80186 interrupt controller will allow interrupts from an external pin regardless of the state of the in-service bit for an interrupt source in order to allow multiple interrupts from a single pin. An in-service bit will continue to be set, however, to inhibit interrupts from other lower-priority 80186 interrupt sources.

Special procedures should be followed when resetting IS bits at the end of interrupt service routines. Software polling of the external master's IS register is required to determine if there is more than one bit set. If so, the IS bit in the 80186 remains active and the next interrupt service routine is entered.

Operation in a Polled Environment

The controller may be used in a polled mode if interrupts are undesirable. When polling, the processor disables interrupts and then polls the interrupt controller whenever it is convenient. Polling the interrupt controller is accomplished by reading the Poll Word (Figure 31). Bit 15 in the poll word indicates to the processor that an interrupt of high enough priority is requesting service. Bits 0–4 indicate to the processor the type vector of the highest-priority source requesting service. Reading the Poll Word causes the In-Service bit of the highest priority source to be set.

It is desirable to be able to read the Poll Word information without guaranteeing service of any pending interrupt, i.e., not set the indicated in-service bit. The 80186 provides a Poll Status Word in addition to the conventional Poll Word to allow this to be done. Poll Word information is duplicated in the Poll Status Word, but reading the Poll Status Word does not set the associated in-service bit. These words are located in two adjacent memory locations in the register file.

Master Mode Features

Programmable Priority

The user can program the interrupt sources into any of eight different priority levels. The programming is done by placing a 3-bit priority level (0–7) in the control register of each interrupt source. (A source with a priority level of 4 has higher priority over all priority levels from 5 to 7. Priority registers containing values lower than 4 have greater priority). All interrupt sources have preprogrammed default priority levels (see Table 4).

If two requests with the same programmed priority level are pending at once, the priority ordering scheme shown in Table 4 is used. If the serviced interrupt routine reenables interrupts, it allows other requests to be serviced.

End-of-Interrupt Command

The end-of-interrupt (EOI) command is used by the programmer to reset the In-Service (IS) bit when an interrupt service routine is completed. The EOI command is issued by writing the proper pattern to the EOI register. There are two types of EOI commands, specific and nonspecific. The nonspecific command does not specify which IS bit is reset. When issued, the interrupt controller automatically resets the IS bit of the highest priority source with an active service routine. A specific EOI command requires that the programmer send the interrupt vector type to the interrupt controller indicating which source's IS bit is to be reset. This command is used when the fully nested structure has been disturbed or the highest priority IS bit that was set does not belong to the service routine in progress.

Trigger Mode

The four external interrupt pins can be programmed in either edge- or level-trigger mode. The control register for each external source has a level-trigger mode (LTM) bit. All interrupt inputs are active HIGH. In the edge sense mode or the level-trigger mode, the interrupt request must remain active (HIGH) until the interrupt request is acknowledged by the 80186 CPU. In the edge-sense mode, if the level remains high after the interrupt is acknowledged, the input is disabled and no further requests will be generated. The input level must go LOW for at least one clock cycle to reenable the input. In the level-trigger mode, no such provision is made: holding the interrupt input HIGH will cause continuous interrupt requests.

Interrupt Vectoring

The 80186 Interrupt Controller will generate interrupt vectors for the integrated DMA channels and the integrated Timers. In addition, the Interrupt Controller will generate interrupt vectors for the external interrupt lines if they are not configured in Cascade or Special Fully Nested Mode. The interrupt vectors generated are fixed and cannot be changed (see Table 4).

Interrupt Controller Registers

The Interrupt Controller register model is shown in Figure 23. It contains 15 registers. All registers can both be read or written unless specified otherwise.

In-Service Register

This register can be read from or written into. The format is shown in Figure 24. It contains the In-Service bit for each of the interrupt sources. The In-Service bit is set to indicate that a source's service routine is in progress. When an In-Service bit is set, the interrupt controller will not generate interrupts to the CPU when it receives interrupt requests from devices with a lower programmed priority level. The TMR bit is the In-Service bit for all three timers; the D0 and D1 bits are the In-Service bits for the two DMA channels; the I0–I3 are the In-Service bits for the external interrupt pins. The IS bit is set when the processor acknowledges an interrupt request either by an interrupt acknowledge or by reading the poll register. The IS bit is reset at the end of the interrupt service routine by an end-of-interrupt command issued by the CPU.

Interrupt Request Register

The internal interrupt sources have interrupt request bits inside the interrupt controller. The format of this register is shown in Figure 24. A read from this register yields the status of these bits. The TMR bit is the logical OR of all timer interrupt requests. D0 and D1 are the interrupt request bits for the DMA channels.

The state of the external interrupt input pins is also indicated. The state of the external interrupt pins is not a stored condition inside the interrupt controller, therefore the external interrupt bits cannot be written. The external interrupt request bits show exactly when an interrupt request is given to the interrupt controller, so if edge-triggered mode is selected, the bit in the register will be HIGH only after an inactive-to-active transition. For internal interrupt sources, the register bits are set when a request arrives and are reset when the processor acknowledges the requests.

Mask Register

This is a 16-bit register that contains a mask bit for each interrupt source. The format for this register is shown in Figure 24. A one in a bit position corresponding to a particular source serves to mask the source from generating interrupts. These mask bits are the exact same bits which are used in the individual control registers; programming a mask bit using the mask register will also change this bit in the individual control registers, and vice versa.

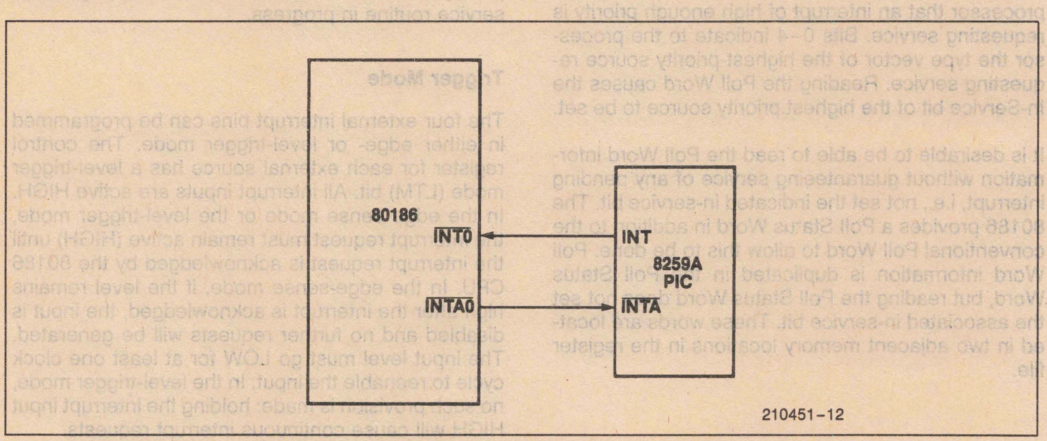


Figure 22. Cascade Mode Interrupt Connection

	OFFSET
INT3 CONTROL REGISTER	3EH
INT2 CONTROL REGISTER	3CH
INT1 CONTROL REGISTER	3AH
INT0 CONTROL REGISTER	38H
DMA 1 CONTROL REGISTER	36H
DMA 0 CONTROL REGISTER	34H
TIMER CONTROL REGISTER	32H
INTERRUPT STATUS REGISTER	30H
INTERRUPT REQUEST REGISTER	2EH
IN-SERVICE REGISTER	2CH
PRIORITY MASK REGISTER	2AH
MASK REGISTER	28H
POLL STATUS REGISTER	26H
POLL REGISTER	24H
EOI REGISTER	22H

Figure 23. Interrupt Controller Registers (Non-iRMX™ 86 Mode)

Priority Mask Register

This register is used to mask all interrupts below particular interrupt priority levels. The format of this register is shown in Figure 25. The code in the lower three bits of this register inhibits interrupts of priority lower (a higher priority number) than the code specified. For example, 100 written into this register masks interrupts of level five (101), six (110), and seven (111). The register is reset to seven (111) upon RESET so all interrupts are unmasked.

Interrupt Status Register

This register contains general interrupt controller status information. The format of this register is shown in Figure 26. The bits in the status register have the following functions:

DHLT: DMA Halt Transfer; setting this bit halts all DMA transfers. It is automatically set whenever a non-maskable interrupt occurs, and it is reset when an IRET instruction is executed. The purpose of this bit is to allow prompt service of all non-maskable interrupts. This bit may also be set by the CPU.

IRTx: These three bits represent the individual timer interrupt request bits. These bits are used to differentiate the timer interrupts, since the timer IR bit in the interrupt request register is the "OR" function of all timer interrupt request. Note that setting any one of these three bits initiates an interrupt request to the interrupt controller.

15	14	•	•	•	10	9	8	7	6	5	4	3	2	1	0
0	0	•	•	•	0	0	0	13	12	I1	I0	D1	D0	0	TMR

Figure 24. In-Service, Interrupt Request, and Mask Register Formats

15	14	•	•	•	•	•	•	•	•	•	3	2	1	0
0	0	•	•	•	•	•	•	•	•	•	0	PRM2	PRM1	PRM0

Figure 25. Priority Mask Register Format

15	14	•	•	•	•	•	7	6	5	4	3	2	1	0
DHLT	0	•	•	•	•	•	0	0	0	0	0	IRT2	IRT1	IRT0

Figure 26. Interrupt Status Register Format

Timer, DMA 0, 1; Control Register

These registers are the control words for all the internal interrupt sources. The format for these registers is shown in Figure 27. The three bit positions PR0, PR1, and PR2 represent the programmable priority level of the interrupt source. The MSK bit inhibits interrupt requests from the interrupt source. The MSK bits in the individual control registers are the exact same bits as are in the Mask Register; modifying them in the individual control registers will also modify them in the Mask Register, and vice versa.

INT0-INT3 Control Registers

These registers are the control words for the four external input pins. Figure 28 shows the format of the INT0 and INT1 Control registers; Figure 29 shows the format of the INT2 and INT3 Control registers. In cascade mode or special fully nested mode, the control words for INT2 and INT3 are not used.

The bits in the various control registers are encoded as follows:

PRO-2: Priority programming information. Highest Priority = 000, Lowest Priority = 111

LTM: Level-trigger mode bit. 1 = level-triggered; 0 = edge-triggered. Interrupt Input levels are active high. In level-triggered mode, an interrupt is generated whenever the external line is high. In edge-triggered mode, an interrupt will be generated only when this level is proceded by an inactive-to-active transition on the line. In both cases, the level must remain active until the interrupt is acknowledged.

MSK: Mask bit, 1 = mask; 0 = non-mask.

C: Cascade mode bit, 1 = cascade; 0 = direct

SFNM: Special fully nested mode bit, 1 = SFNM

EOI Register

The end of the interrupt register is a command register which can only be written into. The format of this register is shown in Figure 30. It initiates an EOI command when written to by the 80186 CPU.

The bits in the EOI register are encoded as follows:

S_x: Encoded information that specifies an interrupt source vector type as shown in Table 4. For example, to reset the In-Service bit for DMA channel 0, these bits should be set to 01010, since the vector type for DMA channel 0 is 10. Note that to reset the single In-Service bit for any of the three timers, the vector type for timer 0 (8) should be written in this register.

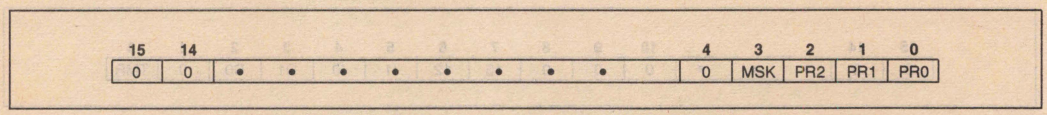


Figure 27. Timer/DMA Control Registers Formats

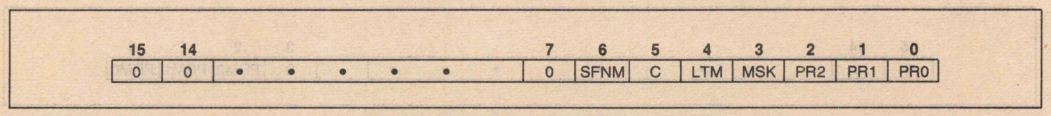


Figure 28. INT0/INT1 Control Register Formats

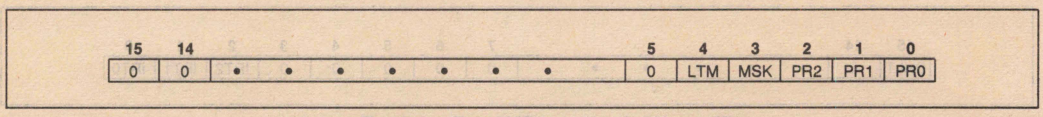


Figure 29. INT2/INT3 Control Register Formats

NSPEC/SPEC: A bit that determines the type of EOI command. Nonspecific = 1, Specific = 0.

Poll and Poll Status Registers

These registers contain polling information. The format of these registers is shown in Figure 31. They can only be read. Reading the Poll register constitutes a software poll. This will set the IS bit of the highest priority pending interrupt. Reading the poll status register will not set the IS bit of the highest priority pending interrupt; only the status of pending interrupts will be provided.

Encoding of the Poll and Poll Status register bits are as follows:

S_X: Encoded information that indicates the vector type of the highest priority interrupting source. Valid only when INTREQ = 1.

INTREQ: This bit determines if an interrupt request is present. Interrupt Request = 1; no Interrupt Request = 0.

iRMX™ 86 COMPATIBILITY MODE

This mode allows iRMX 86-80186 compatibility. The interrupt model of iRMX 86 requires one master and multiple slave 8259As in cascaded fashion. When iRMX mode is used, the internal 80186 interrupt controller will be used as a slave controller to an external master interrupt controller. The internal 80186 resources will be monitored through the internal interrupt controller, while the external controller functions as the system master interrupt controller.

Upon reset, the 80186 interrupt controller will be in the non-iRMX 86 mode of operation. To set the controller in the iRMX 86 mode, bit 14 of the Relocation Register should be set.

Because of pin limitations caused by the need to interface to an external 8259A master, the internal interrupt controller will no longer accept external inputs. There are however, enough 80186 interrupt controller inputs (internally) to dedicate one to each timer. In this mode, each timer interrupt source has its own mask bit, IS bit, and control word.

The iRMX 86 operating system requires peripherals to be assigned fixed priority levels. This is incompatible with the normal operation of the 80186 interrupt controller. Therefore, the initialization software must program the proper priority levels for each source. The required priority levels for the internal interrupt sources in iRMX mode are shown in Table 16.

Table 16. Internal Source Priority Level

Priority Level	Interrupt Source
0	Timer 0
1	(reserved)
2	DMA 0
3	DMA 1
4	Timer 1
5	Timer 2

These level assignments must remain fixed in the iRMX 86 mode of operation.

iRMX™ 86 Mode External Interface

The configuration of the 80186 with respect to an external 8259A master is shown in Figure 32. The INT0 input is used as the 80186 CPU interrupt input. INT3 functions as an output to send the 80186 slave-interrupt-request to one of the 8 master-PIC-inputs.

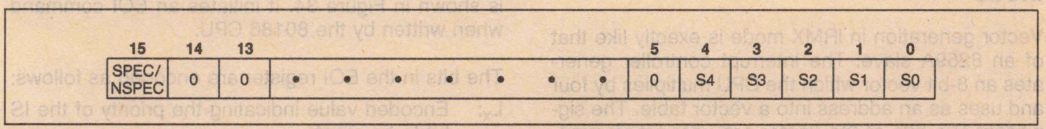


Figure 30. EOI Register Format

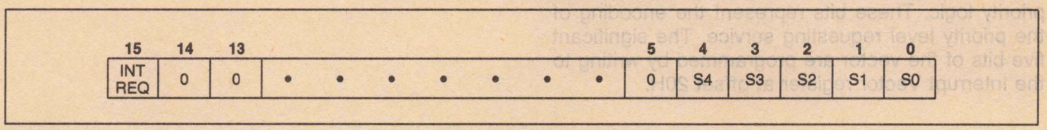


Figure 31. Poll Register Format

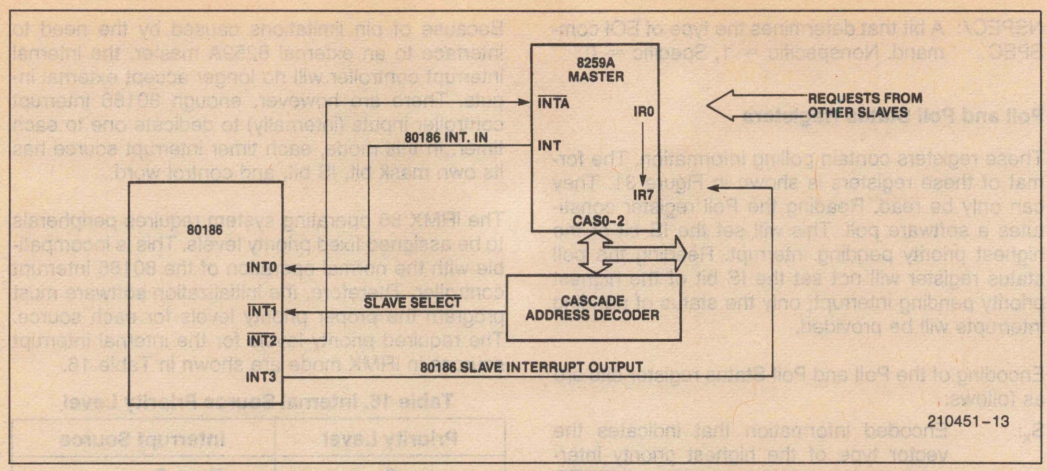


Figure 32. iRMX™ 86 Interrupt Controller Interconnection

Correct master-slave interface requires decoding of the slave addresses (CAS0-2). Slave 8259As do this internally. Because of pin limitations, the 80186 slave address will have to be decoded externally. $\overline{INT1}$ is used as a slave-select input. Note that the slave vector address is transferred internally, but the READY input must be supplied externally.

$\overline{INT2}$ is used as an acknowledge output, suitable to drive the $\overline{INTA}$ input of an 8259A.

Interrupt Nesting

iRMX 86 mode operation allows nesting of interrupt requests. When an interrupt is acknowledged, the priority logic masks off all priority levels except those with equal or higher priority.

Vector Generation in the iRMX™ 86 Mode

Vector generation in iRMX mode is exactly like that of an 8259A slave. The interrupt controller generates an 8-bit vector which the CPU multiplies by four and uses as an address into a vector table. The significant five bits of the vector are user-programmable while the lower three bits are generated by the priority logic. These bits represent the encoding of the priority level requesting service. The significant five bits of the vector are programmed by writing to the Interrupt Vector register at offset 20H.

Specific End-of-Interrupt

In iRMX mode the specific EOI command operates to reset an in-service bit of a specific priority. The user supplies a 3-bit priority-level value that points to an in-service bit to be reset. The command is executed by writing the correct value in the Specific EOI register at offset 22H.

Interrupt Controller Registers in the iRMX™ 86 Mode

All control and command registers are located inside the internal peripheral control block. Figure 33 shows the offsets of these registers.

End-of-Interrupt Register

The end-of-interrupt register is a command register which can only be written. The format of this register is shown in Figure 34. It initiates an EOI command when written by the 80186 CPU.

The bits in the EOI register are encoded as follows:

L_X: Encoded value indicating the priority of the IS bit to be reset.

In-Service Register

This register can be read from or written into. It contains the in-service bit for each of the internal interrupt sources. The format for this register is shown in Figure 35. Bit positions 2 and 3 correspond to the DMA channels; positions 0, 4, and 5 correspond to the integral timers. The source's IS bit is set when the processor acknowledges its interrupt request.

Interrupt Request Register

This register indicates which internal peripherals have interrupt requests pending. The format of this register is shown in Figure 35. The interrupt request bits are set when a request arrives from an internal source, and are reset when the processor acknowledges the request.

Mask Register

The register contains a mask bit for each interrupt source. The format for this register is shown in Figure 35. If the bit in this register corresponding to a particular interrupt source is set, any interrupts from that source will be masked. These mask bits are exactly the same bits which are used in the individual control registers, i.e., changing the state of a mask bit in this register will also change the state of the mask bit in the individual interrupt control register corresponding to the bit.

Control Registers

These registers are the control words for all the internal interrupt sources. The format of these registers is shown in Figure 36. Each of the timers and both of the DMA channels have their own Control Register.

The bits of the Control Registers are encoded as follows:

pr_x: 3-bit encoded field indicating a priority level for the source; note that each source must be programmed at specified levels.

msk: mask bit for the priority level indicated by pr_x bits.

	OFFSET
LEVEL 5 CONTROL REGISTER (TIMER 2)	3AH
LEVEL 4 CONTROL REGISTER (TIMER 1)	38H
LEVEL 3 CONTROL REGISTER (DMA 1)	36H
LEVEL 2 CONTROL REGISTER (DMA 0)	34H
LEVEL 0 CONTROL REGISTER (TIMER 0)	32H
INTERRUPT STATUS REGISTER	30H
INTERRUPT-REQUEST REGISTER	2EH
IN-SERVICE REGISTER	2CH
PRIORITY-LEVEL MASK REGISTER	2AH
MASK REGISTER	28H
SPECIFIC EOI REGISTER	22H
INTERRUPT VECTOR REGISTER	20H

Figure 33. Interrupt Controller Registers (iRMX™ 86 Mode)

15	14	13	•	•	•	•	8	7	6	5	4	3	2	1	0
0	0	0					0	0	0	0	0	0	L2	L1	L0

Figure 34. Specific EOI Register Format

15	14	13	•	•	•	•	8	7	6	5	4	3	2	1	0
0	0	0					0	0	0	TMR2	TMR1	D1	D0	0	TMR0

Figure 35. In-Service, Interrupt Request, and Mask Register Format

Interrupt Vector Register

This register provides the upper five bits of the interrupt vector address. The format of this register is shown in Figure 37. The interrupt controller itself provides the lower three bits of the interrupt vector as determined by the priority level of the interrupt request.

The format of the bits in this register is:

t_x: 5-bit field indicating the upper five bits of the vector address.

Priority-Level Mask Register

This register indicates the lowest priority-level interrupt which will be serviced.

The encoding of the bits in this register is:

m_x: 3-bit encoded field indication priority-level value. All levels of lower priority will be masked.

Interrupt Status Register

This register is defined exactly as in Non-iRMX Mode. (See Fig. 26.)

Interrupt Controller and Reset

Upon RESET, the interrupt controller will perform the following actions:

- All SFNM bits reset to 0, implying Fully Nested Mode.
- All PR bits in the various control registers set to 1. This places all sources at lowest priority (level 111).
- All LTM bits reset to 0, resulting in edge-sense mode.
- All Interrupt Service bits reset to 0.
- All Interrupt Request bits reset to 0.
- All MSK (Interrupt Mask) bits set to 1 (mask).
- All C (Cascade) bits reset to 0 (non-cascade).
- All PRM (Priority Mask) bits set to 1, implying no levels masked.
- Initialized to non-iRMX 86 mode.

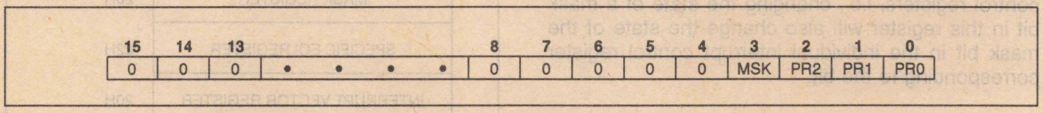


Figure 36. Control Word Format

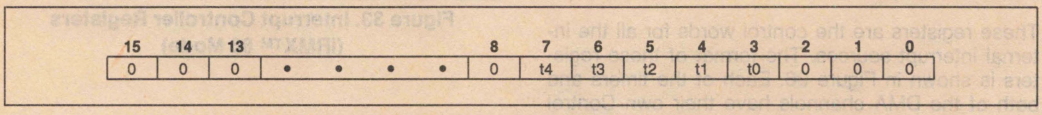


Figure 37. Interrupt Vector Register Format

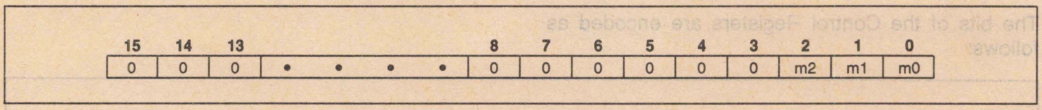


Figure 38. Priority Level Mask Register

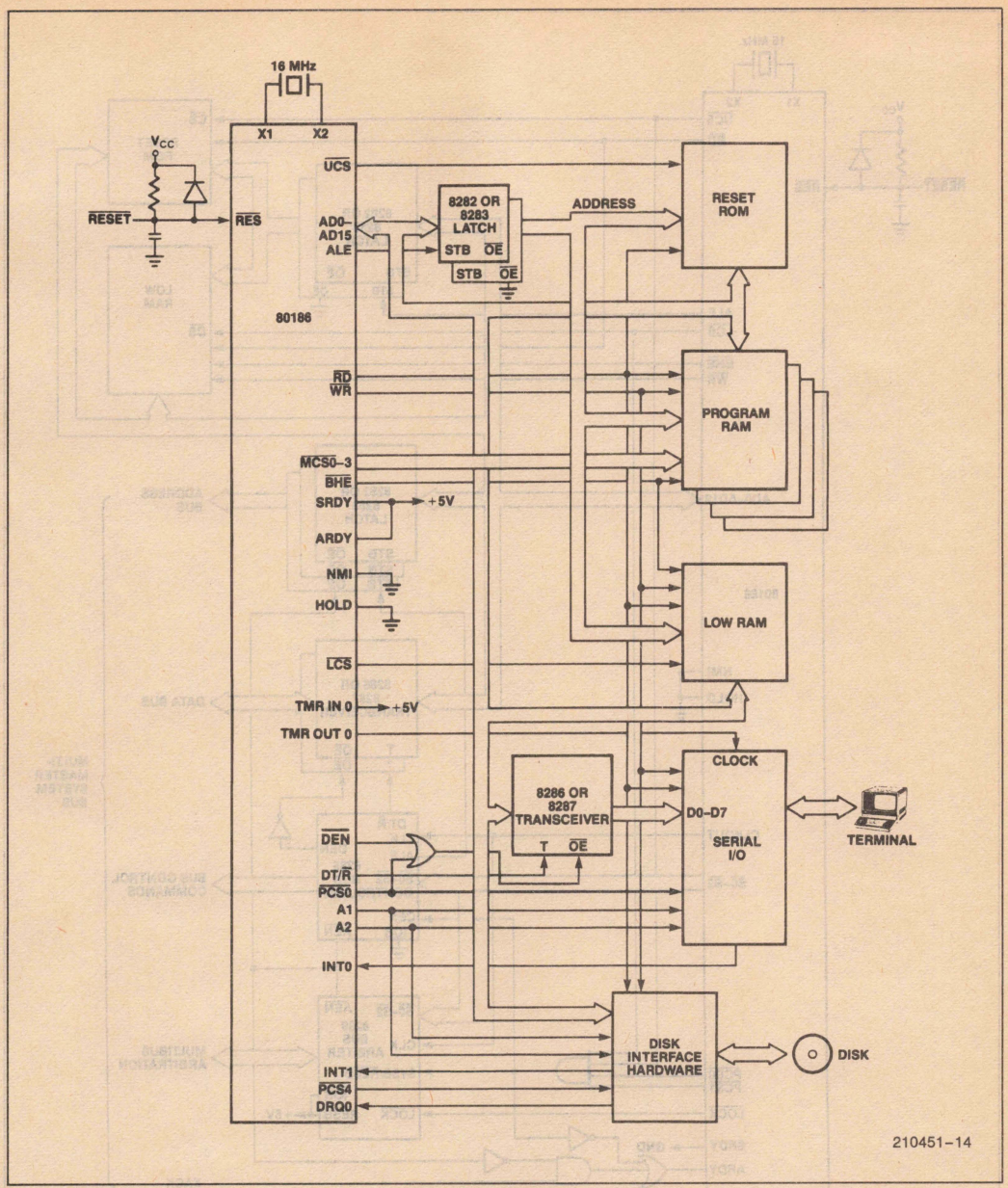


Figure 39. Typical 80186 Computer

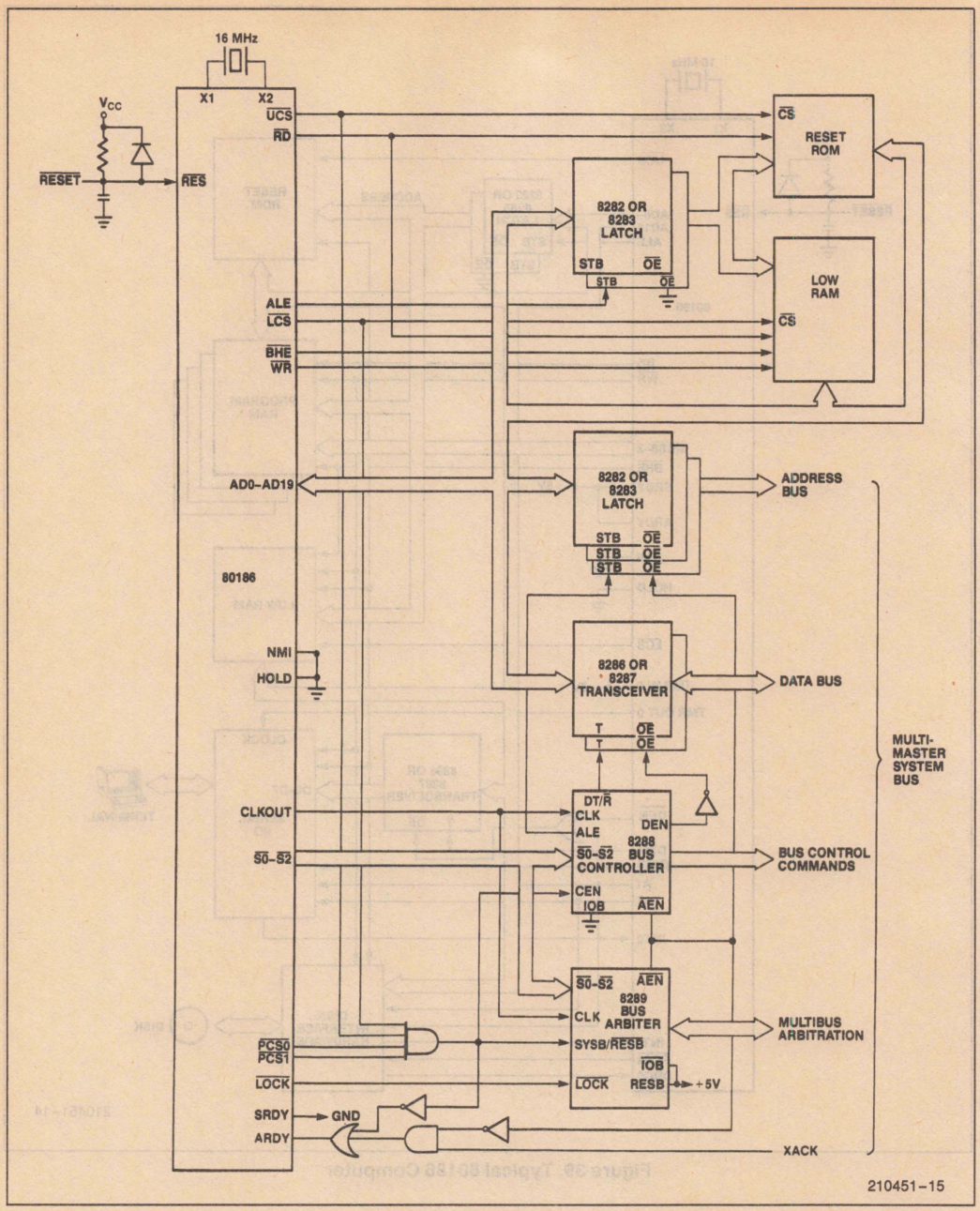


Figure 40. Typical 80186 Multi-Master Bus Interface

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature under Bias .....0°C to 70°C
Case Temperature under Bias .....0°C to +110°C
Storage Temperature .....−65°C to +150°C
Voltage on any Pin with Respect to Ground.....−1.0V to +7V
Power Dissipation .....3 Watt

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS (T_A = 0°C to +70°C, T_C = 0°C to +110°C, V_{CC} = 5V ±10%)

Applicable to 80186 (8 MHz), 80186-10 (10 MHz), and 80186-12 (12.5 MHz)

Symbol	Parameter	Min	Max	Units	Test Conditions
V_{IL}	Input Low Voltage	−0.5	+0.8	Volts	
V_{IH}	Input High Voltage (All except X1 and $\overline{RES}$)	2.0	V_{CC} + 0.5	Volts	
V_{IH1}	Input High Voltage ($\overline{RES}$)	3.0	V_{CC} + 0.5	Volts	
V_{OL}	Output Low Voltage		0.45	Volts	I_a = 2.5 mA for $\overline{S0}$–$\overline{S2}$ I_a = 2.0 mA for all other outputs
V_{OH}	Output High Voltage	2.4		Volts	I_{oa} = −400 μA
I_{CC}	Power Supply Current		600*	mA	T_A = −40°C
			550	mA	T_A = 0°C
			415	mA	T_A = +70°C
I_{LI}	Input Leakage Current		±10	μA	0V < V_{IN} < V_{CC}
I_{LO}	Output Leakage Current		±10	μA	0.45V < V_{OUT} < V_{CC}
V_{CLO}	Clock Output Low		0.6	Volts	I_a = 4.0 mA
V_{CHO}	Clock Output High	4.0		Volts	I_{oa} = −200 μA
V_{CLI}	Clock Input Low Voltage	−0.5	0.6	Volts	
V_{CHI}	Clock Input High Voltage	3.9	V_{CC} + 1.0	Volts	
C_{IN}	Input Capacitance		10	pF	
C_{IO}	I/O Capacitance		20	pF	

*For extended temperature parts only.

PIN TIMINGS

A.C. CHARACTERISTICS (T_A = 0°C to +70°C, T_C = 0°C to +110°C, V_{CC} = 5V ±10%)

80186 Timing Requirements All Timings Measured At 1.5 Volts Unless Otherwise Noted.

Symbol	Parameter	80186 (8 MHz)		80186-10 (10 MHz)		80186-12 (12.5 MHz) Preliminary		Units	Test Conditions
		Min	Max	Min	Max	Min	Max		
T_{DVCL}	Data in Setup (A/D)	20		15		15		ns	
T_{CLDX}	Data in Hold (A/D)	10		8		8		ns	
T_{ARYHCH}	Asynchronous Ready (AREADY) active setup time*	20		15		15		ns	
T_{ARYLCL}	AREADY inactive setup time	35		25		25		ns	
T_{CHARYX}	AREADY hold time	15		15		15		ns	
T_{ARYCHL}	Asynchronous Ready inactive hold time	15		15		15		ns	
T_{SRYCL}	Synchronous Ready (SREADY) transition setup time	20		20		20		ns	
T_{CLSRY}	SREADY transition hold time	15		15		15		ns	
T_{HVCL}	HOLD Setup*	25		20		20		ns	
T_{INVCH}	INTR, NMI, $\overline{TEST}$, TIMERIN, Setup*	25		25		25		ns	
T_{INVCL}	DRQ0, DRQ1, Setup*	25		20		20		ns	

80186 Master Interface Timing Responses

T_{CLAV}	Address Valid Delay	5	55	5	44	4	33	ns	C_L = 20–200 pF all outputs (except T_{CLTMV}) @ 8 & 10 MHz C_L = 20–100 pF all outputs @ 12.5 MHz
T_{CLAX}	Address Hold	10		10		8		ns	
T_{CLAZ}	Address Float Delay	T_{CLAX}	35	T_{CLAX}	30	8	25	ns	
T_{CHCZ}	Command Lines Float Delay		45		40		33	ns	
T_{CHCV}	Command Lines Valid Delay (after float)		55		45		37	ns	
T_{LHLL}	ALE Width	$T_{CLCL}-35$		$T_{CLCL}-30$		$T_{CLCL}-30$		ns	
T_{CHLH}	ALE Active Delay		35		30		25	ns	
T_{CHLL}	ALE Inactive Delay		35		30		25	ns	
T_{LLAX}	Address Hold to ALE Inactive	$T_{CHCL}-25$		$T_{CHCL}-20$		$T_{CHCL}-15$		ns	
T_{CLDV}	Data Valid Delay	10	44	10	40	8	33	ns	
T_{CLDOX}	Data Hold Time	10		10		8		ns	
T_{WHDX}	Data Hold after WR	$T_{CLCL}-40$		$T_{CLCL}-34$		$T_{CLCL}-20$		ns	
T_{CVCTV}	Control Active Delay 1	5	50	5	40	5	47	ns	
T_{CHCTV}	Control Active Delay 2	10	55	10	44	8	37	ns	
T_{CVCTX}	Control Inactive Delay	5	55	5	44	4	37	ns	
T_{CVDEX}	$\overline{DEN}$ Inactive Delay (Non-Write Cycle)	10	70	10	56	8	47	ns	

*To guarantee recognition at next clock.

PIN TIMINGS (Continued)

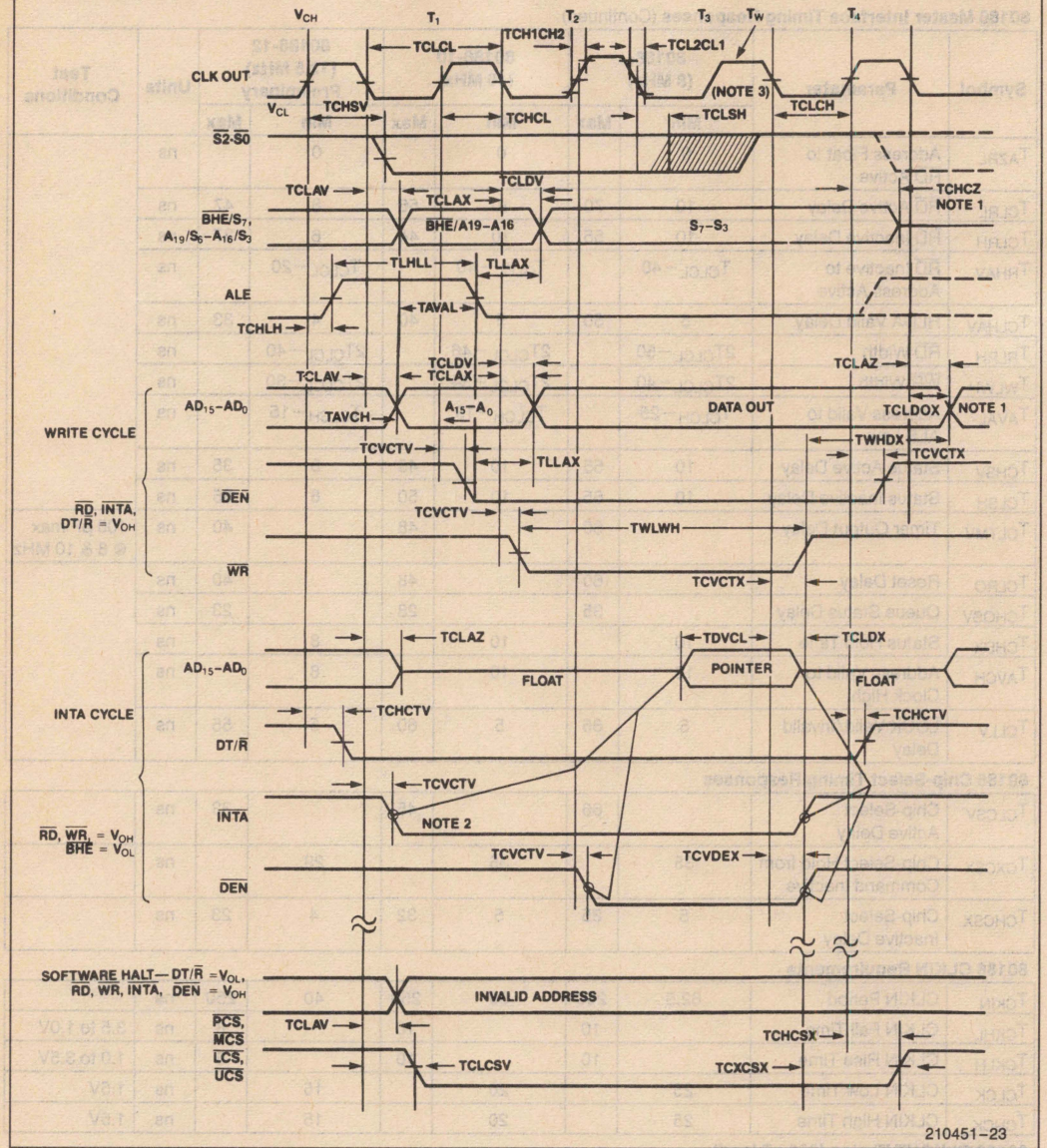
A.C. CHARACTERISTICS (T_A = 0°C to +70°C, T_C = 0°C to +110°C, V_{CC} = 5V ±10%) (Continued)

80186 Master Interface Timing Responses (Continued)

Symbol	Parameter	80186 (8 MHz)		80186-10 (10 MHz)		80186-12 (12.5 MHz) Preliminary		Units	Test Conditions
		Min	Max	Min	Max	Min	Max		
T_{AZRL}	Address Float to $\overline{RD}$ Active	0		0		0		ns	
T_{CLRL}	$\overline{RD}$ Active Delay	10	70	10	56	8	47	ns	
T_{CLRH}	$\overline{RD}$ Inactive Delay	10	55	10	44	8	37	ns	
T_{RHAV}	$\overline{RD}$ Inactive to Address Active	$T_{CLCL}-40$		$T_{CLCL}-40$		$T_{CLCL}-20$		ns	
T_{CLHAV}	HLDA Valid Delay	5	50	5	40	4	33	ns	
T_{RLRH}	$\overline{RD}$ Width	$2T_{CLCL}-50$		$2T_{CLCL}-46$		$2T_{CLCL}-40$		ns	
T_{WLWH}	$\overline{WR}$ Width	$2T_{CLCL}-40$		$2T_{CLCL}-34$		$2T_{CLCL}-30$		ns	
T_{AVAL}	Address Valid to ALE Low	$T_{CLCH}-25$		$T_{CLCH}-19$		$T_{CLCH}-15$		ns	
T_{CHSV}	Status Active Delay	10	55	10	45	8	35	ns	
T_{CLSH}	Status Inactive Delay	10	65	10	50	8	35	ns	
T_{CLTMV}	Timer Output Delay		60		48		40	ns	100 pF max @ 8 & 10 MHz
T_{CLRO}	Reset Delay		60		48		40	ns	
T_{CHQSV}	Queue Status Delay		35		28		23	ns	
T_{CHDX}	Status Hold Time	10		10		8		ns	
T_{AVCH}	Address Valid to Clock High	10		10		8		ns	
T_{CLLV}	$\overline{LOCK}$ Valid/Invalid Delay	5	65	5	60	5	55	ns	
80186 Chip-Select Timing Responses									
T_{CLCSV}	Chip-Select Active Delay		66		45		33	ns	
T_{CXCSX}	Chip-Select Hold from Command Inactive	35		35		29		ns	
T_{CHCSX}	Chip-Select Inactive Delay	5	35	5	32	4	23	ns	
80186 CLKIN Requirements									
T_{CKIN}	CLKIN Period	62.5	250	50	250	40	250	ns	
T_{CKHL}	CLKIN Fall Time		10		10		8	ns	3.5 to 1.0V
T_{CKLH}	CLKIN Rise Time		10		10		8	ns	1.0 to 3.5V
T_{CLCK}	CLKIN Low Time	25		20		15		ns	1.5V
T_{CHCK}	CLKIN High Time	25		20		15		ns	1.5V
80186 CLKOUT Timing (200 pF load)									
T_{CICO}	CLKIN to CLKOUT Skew		50		25		21	ns	
T_{CLCL}	CLKOUT Peiod	125	500	100	500	80	500	ns	
T_{CLCH}	CLKOUT Low Time	$\frac{1}{2}T_{CLCL}-7.5$		$\frac{1}{2}T_{CLCL}-6.0$		$\frac{1}{2}T_{CLCL}-6.0$		ns	1.5V
T_{CHCL}	CLKOUT High Time	$\frac{1}{2}T_{CLCL}-7.5$		$\frac{1}{2}T_{CLCL}-6.0$		$\frac{1}{2}T_{CLCL}-6.0$		ns	1.5V
T_{CH1CH2}	CLKOUT Rise Time		15		12		10	ns	1.0 to 3.5V
T_{CL2CL1}	CLKOUT Fall Time		15		12		10	ns	3.5 to 1.0V

WAVEFORMS

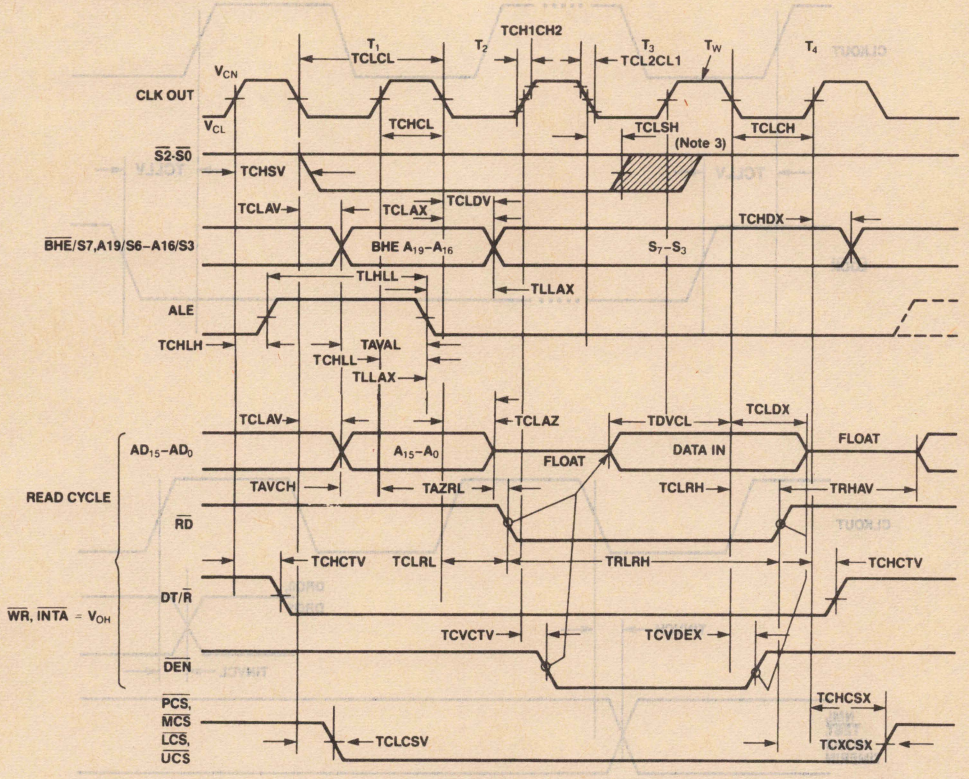
MAJOR CYCLE TIMING



210451-23

WAVEFORMS (Continued)

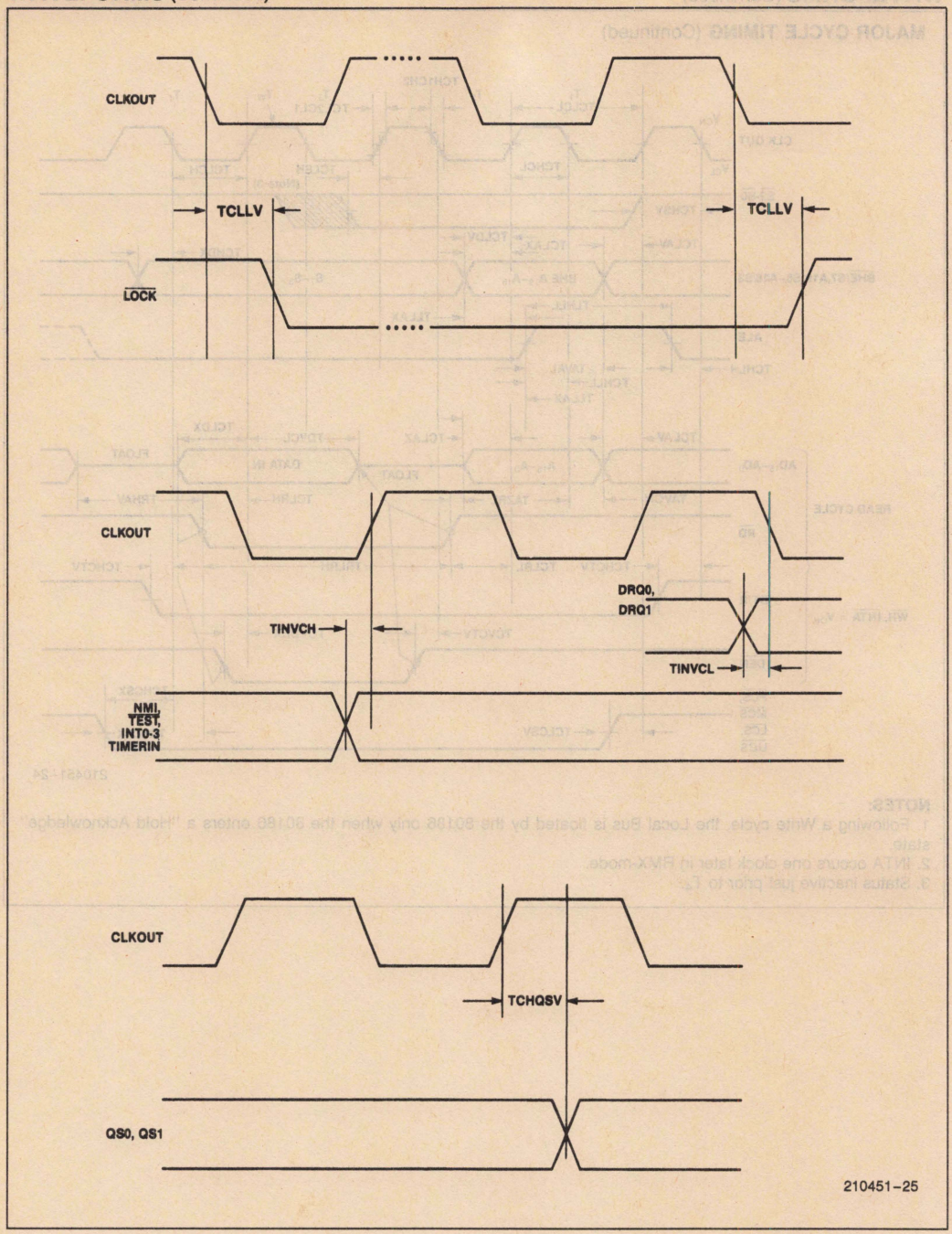
MAJOR CYCLE TIMING (Continued)



210451-24

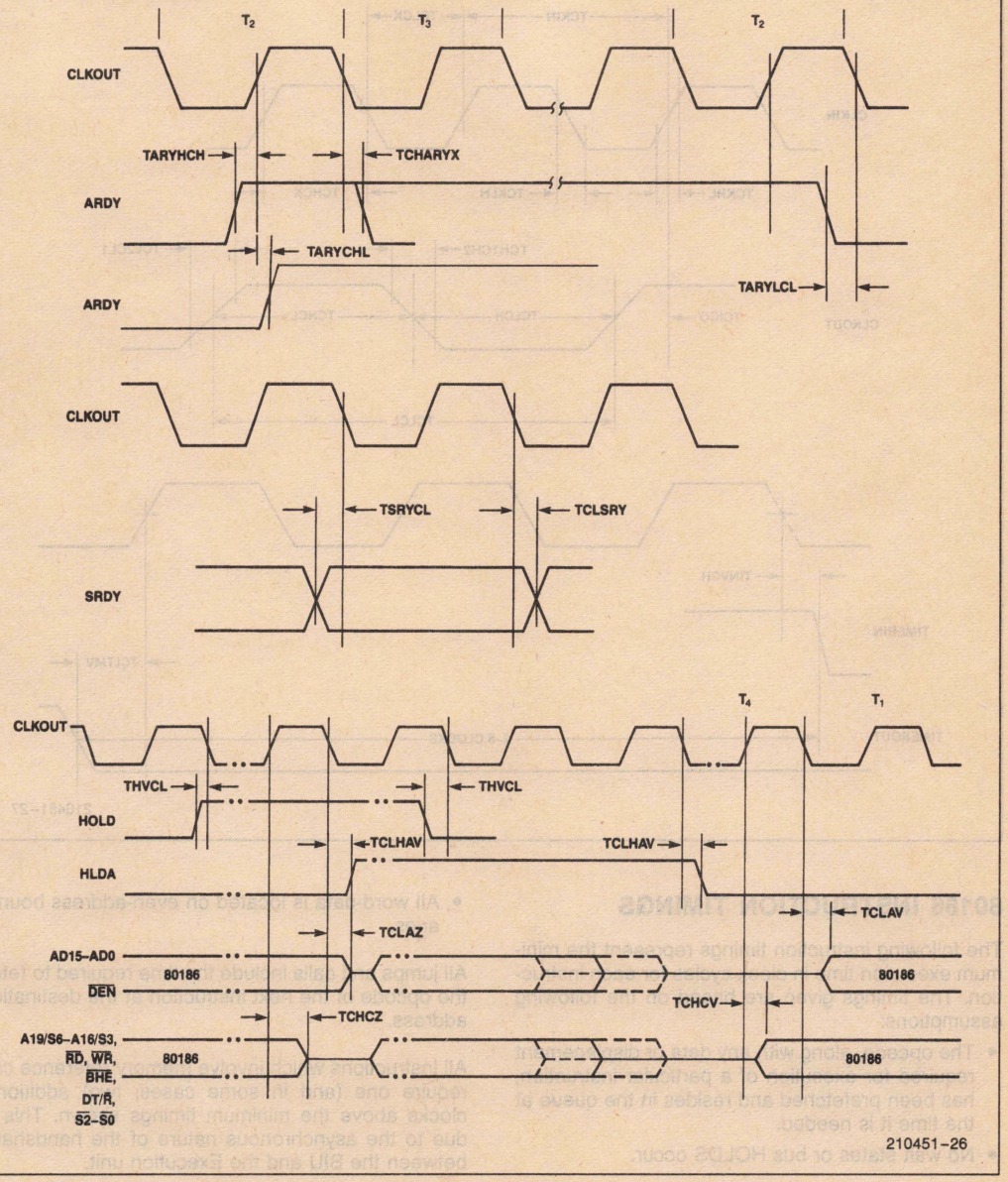
NOTES:
1. Following a Write cycle, the Local Bus is floated by the 80186 only when the 80186 enters a "Hold Acknowledge" state.
2. INTA occurs one clock later in RMX-mode.
3. Status inactive just prior to T_4.

WAVEFORMS (Continued)



WAVEFORMS (Continued)

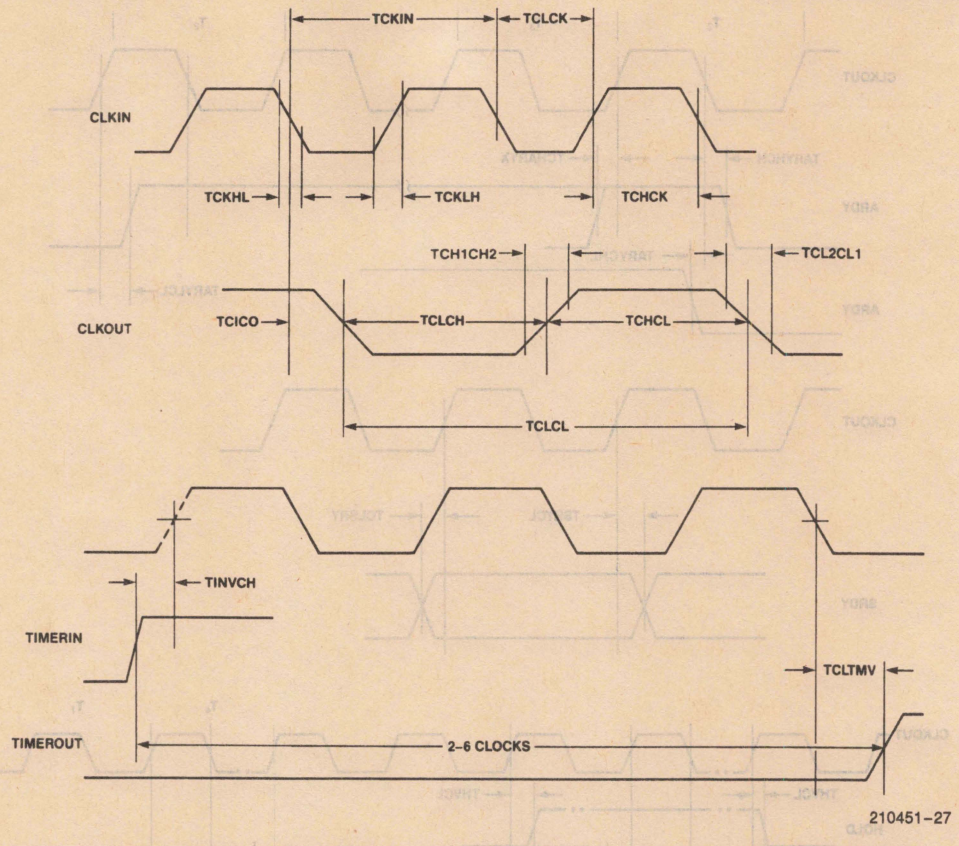
HOLD-HLDA TIMING



210451-26

WAVEFORMS (Continued)

TIMER ON 80186



80186 INSTRUCTION TIMINGS

The following instruction timings represent the minimum execution time in clock cycles for each instruction. The timings given are based on the following assumptions:

- The opcode, along with any data or displacement required for execution of a particular instruction, has been prefetched and resides in the queue at the time it is needed.
- No wait states or bus HOLDS occur.
- All word-data is located on even-address boundaries.

All jumps and calls include the time required to fetch the opcode of the next instruction at the destination address.

All instructions which involve memory reference can require one (and in some cases, two) additional clocks above the minimum timings shown. This is due to the asynchronous nature of the handshake between the BIU and the Execution unit.

INSTRUCTION SET SUMMARY

Function	Format	Clock Cycles	Comments
DATA TRANSFER			
MOV = Move:			
Register to Register/Memory	1000100w mod reg r/m	2/12	
Register/memory to register	1000101w mod reg r/m	2/9	
Immediate to register/memory	1100011w mod 000 r/m data data if w=1	12–13	8/16-bit
Immediate to register	1011w reg data data if w=1	3–4	8/16-bit
Memory to accumulator	1010000w addr-low addr-high	9	
Accumulator to memory	1010001w addr-low addr-high	8	
Register/memory to segment register	10001110 mod 0 reg r/m	2/9	
Segment register to register/memory	10001100 mod 0 reg r/m	2/11	
PUSH = Push:			
Memory	11111111 mod 1 1 0 r/m	16	
Register	01010 reg	10	
Segment register	000 reg 1 1 0	9	
Immediate	011010s0 data data if s=0	10	
PUSHA = Push All	01100000	36	
POP = Pop:			
Memory	10001111 mod 0 0 0 r/m	20	
Register	01011 reg	10	
Segment register	0 0 0 reg 1 1 1 (reg≠01)	8	
POPA = Pop All	01100001	51	
XCHG = Exchange:			
Register/memory with register	1000011w mod reg r/m	4/17	
Register with accumulator	10010 reg	3	
IN = Input from:			
Fixed port	1110010w port	10	
Variable port	1110110w	8	
OUT = Output to:			
Fixed port	1110011w port	9	
Variable port	1110111w	7	
XLAT = Translate byte to AL	11010111	11	
LEA = Load EA to register	10001101 mod reg r/m	6	
LDS = Load pointer to DS	11000101 mod reg r/m (mod≠11)	18	
LES = Load pointer to ES	11000100 mod reg r/m (mod≠11)	18	
LAHF = Load AH with flags	10011111	2	
SAHF = Store AH into flags	10011110	3	
PUSHF = Push flags	10011100	9	
POPF = Pop flags	10011101	8	

Shaded areas indicate instructions not available in 8086, 8088 microsystems.

INSTRUCTION SET SUMMARY (Continued)

Function	Format				Clock Cycles	Comments
DATA TRANSFER (Continued)						
SEGMENT = Segment Override:						
CS	00101110				2	
SS	00110110				2	
DS	00111110				2	
ES	00100110				2	
ARITHMETIC						
ADD = Add:						
Reg/memory with register to either	000000dw	mod reg r/m			3/10	
Immediate to register/memory	100000sw	mod 000 r/m	data	data if s w=01	4/16	
Immediate to accumulator	0000010w	data	data if w=1		3/4	8/16-bit
ADC = Add with carry:						
Reg/memory with register to either	000100dw	mod reg r/m			3/10	
Immediate to register/memory	100000sw	mod 010 r/m	data	data if s w=01	4/16	
Immediate to accumulator	0001010w	data	data if w=1		3/4	8/16-bit
INC = Increment:						
Register/memory	1111111w	mod 000 r/m			3/15	
Register	01000 reg				3	
SUB = Subtract:						
Reg/memory and register to either	001010dw	mod reg r/m			3/10	
Immediate from register/memory	100000sw	mod 101 r/m	data	data if s w=01	4/16	
Immediate from accumulator	0010110w	data	data if w=1		3/4	8/16-bit
SBB = Subtract with borrow:						
Reg/memory and register to either	000110dw	mod reg r/m			3/10	
Immediate from register/memory	100000sw	mod 011 r/m	data	data if s w=01	4/16	
Immediate from accumulator	0001110w	data	data if w=1		3/4	8/16-bit
DEC = Decrement						
Register/memory	1111111w	mod 001 r/m			3/15	
Register	01001 reg				3	
CMP = Compare:						
Register/memory with register	0011101w	mod reg r/m			3/10	
Register with register/memory	0011100w	mod reg r/m			3/10	
Immediate with register/memory	100000sw	mod 111 r/m	data	data if s w=01	3/10	
Immediate with accumulator	0011110w	data	data if w=1		3/4	8/16-bit
NEG = Change sign register/memory	1111011w	mod 011 r/m			3/10	
AAA = ASCII adjust for add	00110111				8	
DAA = Decimal adjust for add	00100111				4	
AAS = ASCII adjust for subtract	00111111				7	
DAS = Decimal adjust for subtract	00101111				4	
MUL = Multiply (unsigned):	1111011w	mod 100 r/m				
Register-Byte					26–28	
Register-Word					35–37	
Memory-Byte					32–34	
Memory-Word					41–43	

Shaded areas indicate instructions not available in 8086, 8088 microsystems.

INSTRUCTION SET SUMMARY (Continued)

Function	Format	Clock Cycles	Comments
ARITHMETIC (Continued)			
IMUL = Integer multiply (signed):	1111011w | mod101 r/m		
Register-Byte		25–28	
Register-Word		34–37	
Memory-Byte		31–34	
Memory-Word		40–43	
IMUL = Integer immediate multiply (signed)	011010s1 | mod reg r/m | data | data if s=0	22–25/29–32	
DIV = Divide (unsigned):	1111011w | mod110 r/m		
Register-Byte		29	
Register-Word		38	
Memory-Byte		35	
Memory-Word		44	
IDIV = Integer divide (signed):	1111011w | mod111 r/m		
Register-Byte		44–52	
Register-Word		53–61	
Memory-Byte		50–58	
Memory-Word		59–67	
AAM = ASCII adjust for multiply	11010100 | 00001010	19	
AAD = ASCII adjust for divide	11010101 | 00001010	15	
CBW = Convert byte to word	10011000	2	
CWD = Convert word to double word	10011001	4	
LOGIC			
Shift/Rotate Instructions:			
Register/Memory by 1	1101000w | mod TTT r/m	2/15	
Register/Memory by CL	1101001w | mod TTT r/m	5+n/17+n	
Register/Memory by Count	1100000w | mod TTT r/m | count	5+n/17+n	
	TTT Instruction 000 ROL 001 ROR 010 RCL 011 RCR 100 SHL/SAL 101 SHR 111 SAR		
AND = **And:**			
Reg/memory and register to either	001000dw | mod reg r/m	3/10	
Immediate to register/memory	1000000w | mod100 r/m | data | data if w=1	4/16	
Immediate to accumulator	0010010w | data | data if w=1	3/4	8/16-bit
TEST = **And function to flags, no result:**			
Register/memory and register	1000010w | mod reg r/m	3/10	
Immediate data and register/memory	1111011w | mod000 r/m | data | data if w=1	4/10	
Immediate data and accumulator	1010100w | data | data if w=1	3/4	8/16-bit
OR = **Or:**			
Reg/memory and register to either	000010dw | mod reg r/m	3/10	
Immediate to register/memory	1000000w | mod001 r/m | data | data if w=1	4/16	
Immediate to accumulator	0000110w | data | data if w=1	3/4	8/16-bit

Shaded areas indicate instructions not available in 8086, 8088 microsystems.

INSTRUCTION SET SUMMARY (Continued)

Function	Format	Clock Cycles	Comments
LOGIC (Continued)			
XOR = Exclusive or:			
Reg/memory and register to either	001100dw | mod reg r/m	3/10	
Immediate to register/memory	1000000w | mod 110 r/m | data | data if w=1	4/16	
Immediate to accumulator	0011010w | data | data if w=1	3/4	8/16-bit
NOT = Invert register/memory	1111011w | mod 010 r/m	3/10	
STRING MANIPULATION			
MOVS = Move byte/word	1010010w	14	
CMPS = Compare byte/word	1010011w	22	
SCAS = Scan byte/word	1010111w	15	
LODS = Load byte/wd to ALAX	1010110w	12	
STOS = Stor byte/wd from ALA	1010101w	10	
INS = Input byte/wd from DX port	0110110w	14	
OUTS = Output byte/wd to DX port	0110111w	14	
Repeated by count in CX			
MOVS = Move string	11110010 | 1010010w	8+8n	
CMPS = Compare string	1111001z | 1010011w	5+22n	
SCAS = Scan string	1111001z | 1010111w	5+15n	
LODS = Load string	11110010 | 1010110w	6+11n	
STOS = Store string	11110010 | 1010101w	6+9n	
INS = Input string	11110010 | 0110110w	8+8n	
OUTS = Output string	11110010 | 0110111w	8+8n	
CONTROL TRANSFER			
CALL = Call:			
Direct within segment	11101000 | disp-low | disp-high	15	
Register/memory indirect within segment	11111111 | mod 010 r/m	13/19	
Direct intersegment	10011010 | segment offset | segment selector	23	
Indirect intersegment	11111111 | mod 011 r/m	(mod ≠ 11)	38
JMP = Unconditional jump:			
Short/long	11101011 | disp-low	14	
Direct within segment	11101001 | disp-low | disp-high	14	
Register/memory indirect within segment	11111111 | mod 100 r/m	11/17	
Direct intersegment	11101010 | segment offset | segment selector	14	
Indirect intersegment	11111111 | mod 101 r/m	(mod ≠ 11)	26

Shaded areas indicate instructions not available in 8086, 8088 microsystems.

INSTRUCTION SET SUMMARY (Continued)

Function	Format				Clock Cycles	Comments
CONTROL TRANSFER (Continued)						
RET = Return from CALL:						
Within segment	11000011				16	
Within seg adding immed to SP	11000010	data-low	data-high		18	
Intersegment	11001011				22	
Intersegment adding immediate to SP	11001010	data-low	data-high		25	
JE/JZ = Jump on equal/zero	01110100	disp			4/13	JMP not taken/JMP taken
JL/JNGE = Jump on less/not greater or equal	01111100	disp			4/13	
JLE/JNG = Jump on less or equal/not greater	01111110	disp			4/13	
JB/JNAE = Jump on below/not above or equal	01110010	disp			4/13	
JBE/JNA = Jump on below or equal/not above	01110110	disp			4/13	
JP/JPE = Jump on parity/parity even	01111010	disp			4/13	
JO = Jump on overflow	01110000	disp			4/13	
JS = Jump on sign	01111000	disp			4/13	
JNE/JNZ = Jump on not equal/not zero	01110101	disp			4/13	
JNL/JGE = Jump on not less/greater or equal	01111101	disp			4/13	
JNLE/JG = Jump on not less or equal/greater	01111111	disp			4/13	
JNB/JAE = Jump on not below/above or equal	01110011	disp			4/13	
JNBE/JA = Jump on not below or equal/above	01110111	disp			4/13	
JNP/JPO = Jump on not par/par odd	01111011	disp			4/13	
JNO = Jump on not overflow	01110001	disp			4/13	
JNS = Jump on not sign	01111001	disp			4/13	
JCXZ = Jump on CX zero	11100011	disp			5/15	
LOOP = Loop CX times	11100010	disp			6/16	LOOP not taken/LOOP taken
LOOPZ/LOOPE = Loop while zero/equal	11100001	disp			6/16	
LOOPNZ/LOOPNE = Loop while not zero/equal	11100000	disp			6/16	
ENTER = Enter Procedure	11001000	data-low	data-high	L		
L = 0					15	
L = 1					25	
L > 1					22+16(n−1)	
LEAVE = Leave Procedure	11001001				8	
INT = Interrupt:						
Type specified	11001101	type			47	
Type 3	11001100				45	if INT. taken/ if INT. not taken
INTO = Interrupt on overflow	11001110				48/4	
IRET = Interrupt return	11001111				28	
BOUND = Detect value out of range	01100010	mod reg r/m			33–35	

Shaded areas indicate instructions not available in 8086, 8088 microsystems.

INSTRUCTION SET SUMMARY (Continued)

Function	Format	Clock Cycles	Comments
PROCESSOR CONTROL			
CLC = Clear carry	11111000	2	
CMC = Complement carry	11110101	2	
STC = Set carry	11111001	2	
CLD = Clear direction	11111100	2	
STD = Set direction	11111101	2	
CLI = Clear interrupt	11111010	2	
STI = Set interrupt	11111011	2	
HLT = Halt	11110100	2	
WAIT = Wait	10011011	6	if $\overline{\text{test}}$ = 0
LOCK = Bus lock prefix	11110000	2	
ESC = Processor Extension Escape	11011TTT mod LLL r/m (TTT LLL are opcode to processor extension)	6	

Shaded areas indicate instructions not available in 8086, 8088 microsystems.

80188
HIGH INTEGRATION 8-BIT MICROPROCESSOR

- **Integrated Feature Set**
  - — Enhanced 8088-2 CPU
  - — Clock Generator
  - — 2 Independent, High-Speed DMA Channels
  - — Programmable Interrupt Controller
  - — 3 Programmable 16-Bit Timers
  - — Programmable Memory and Peripheral Chip-Select Logic
  - — Programmable Wait State Generator
  - — Local Bus Controller
- **Available in 10 MHz (80188-10) and 8 MHz (80188) Versions**
- **High-Performance 8 MHz Processor**
  - — At 8 MHz Provides 2 Times the Performance of the Standard 80188
  - — 2 MByte/Sec Bus Bandwidth Interface @8 MHz
  - — 2.5 MByte/Sec Bus Bandwidth Interface @ 10 MHz
- **Available in EXPRESS**
  - — Standard Temperature with Burn-In
  - — Extended Temperature Range (−40°C to +85°C)
- **8-Bit Data Bus Interface; 16-Bit Internal Architecture**
- **Completely Object Code Compatible with All Existing iAPX 86, 88 Software**
  - — 10 New Instruction Types
- **DRAM Refresh Capability via DMA Channel and Timer 2**
- **Direct Addressing Capability to 1 MByte of Memory and 64 KByte I/O**
- **Complete System Development Support**
  - — Development Software; Assembler, PL/M, Pascal, Fortran, and System Utilities
  - — In-Circuit-Emulator (ICE™-188)
- **High Performance Numerical Coprocessing Capability Through 8087 Interface**
- **Available in 68 Pin:**
  - — Ceramic Leadless Chip Carrier (LCC)
  - — Ceramic Pin Grid Array (PGA)
  - — Plastic Leaded Chip Carrier (PLCC)

(See Packaging Spec., Order #231369)

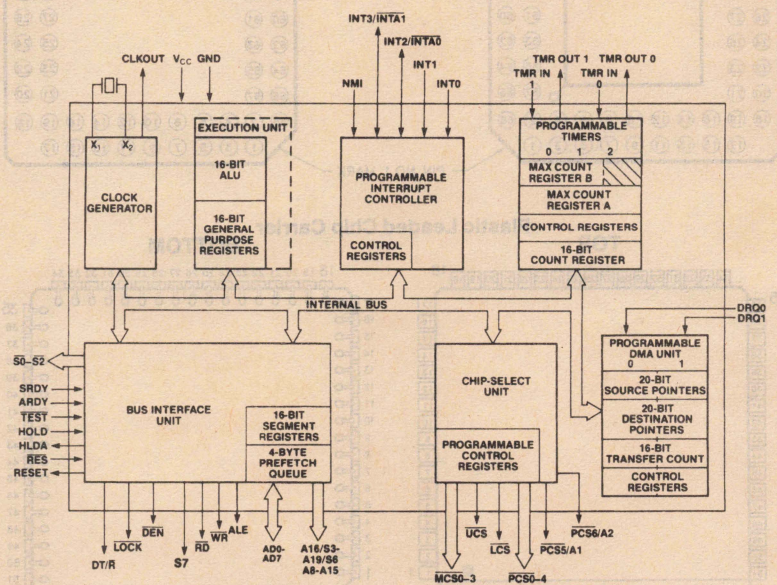


Figure 1. 80188 Block Diagram

October 1986
Order Number: 210706-008

The Intel 80188 is a highly integrated microprocessor with an 8-bit data bus interface and a 16-bit internal architecture to give high performance. The 80188 effectively combines 15–20 of the most common 8088 system components onto one. The 80188 provides two times greater throughput than the standard 5 MHz 8088. The 80188 is upward compatible with 8086 and 8088 software and adds 10 new instruction types to the existing set.

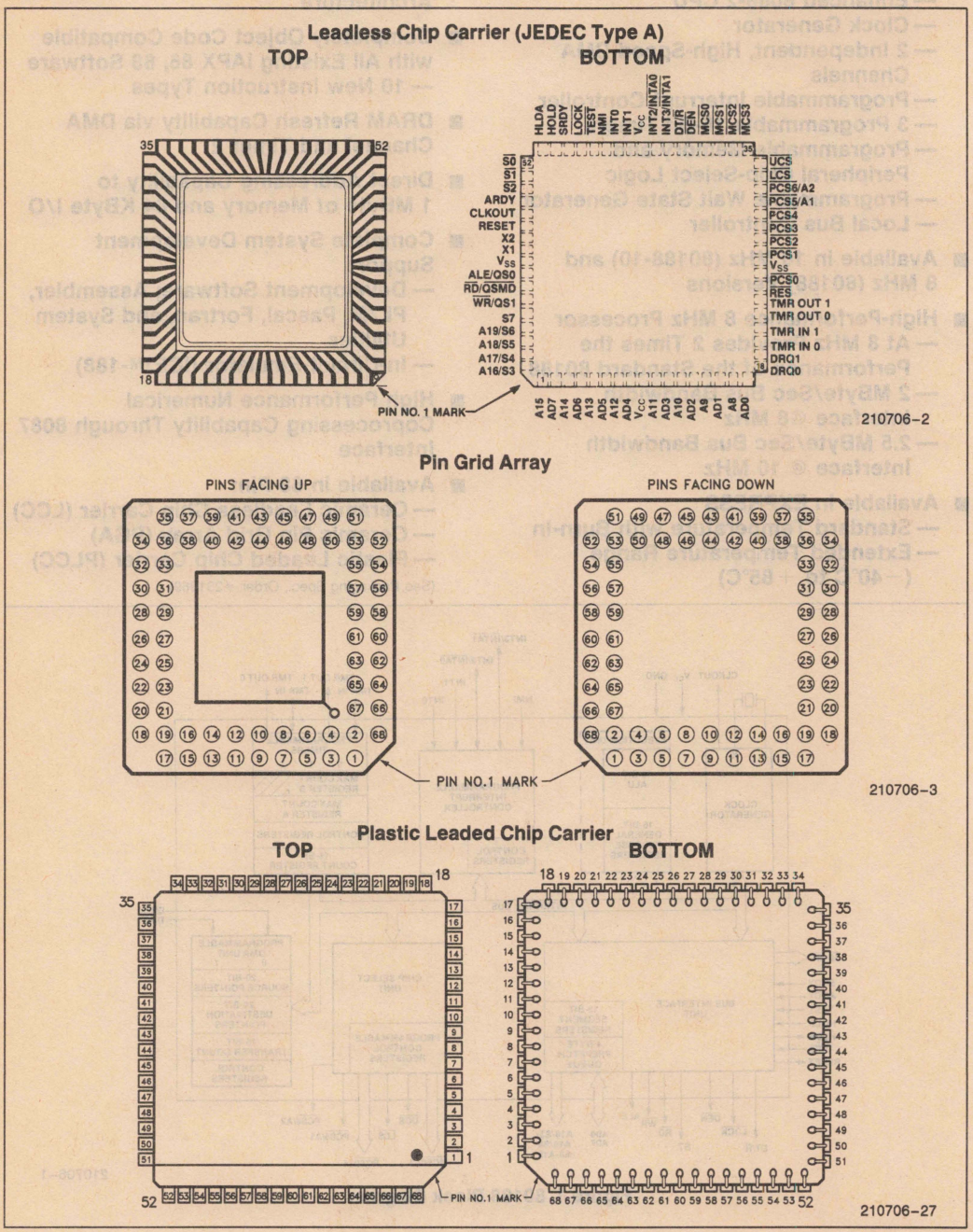


Figure 2. 80188 Pinout Diagram

Table 1. 80188 Pin Description

Symbol	Pin No.	Type	Name and Function
V_{CC}, V_{CC}	9, 43	I	**SYSTEM POWER:** +5 volt power supply.
V_{SS}, V_{SS}	26, 60	I	**SYSTEM GROUND**
RESET	57	O	**RESET OUTPUT:** Indicates that the 80188 CPU is being reset, and can be used as a system reset. It is active HIGH, synchronized with the processor clock, and lasts an integer number of clock periods corresponding to the length of the $\overline{RES}$ signal.
X1, X2	59, 58	I	**CRYSTAL INPUTS:** X1 and X2 provide an external connection for a fundamental mode parallel resonant crystal for the internal crystal oscillator. X1 can interface to an external clock instead of a crystal. In this case, minimize the capacitance on X2 or drive X2 with complemented X1. The input or oscillator frequency is internally divided by two to generate the clock signal (CLKOUT).
CLKOUT	56	O	**CLOCK OUTPUT:** Provides the system with a 50% duty cycle waveform. All device pin timings are specified relative to CLKOUT. CLKOUT has sufficient MOS drive capabilities for the 8087 Numeric Processor Extension.
$\overline{RES}$	24	I	**SYSTEM RESET:** Causes the 80188 to immediately terminate its present activity, clear the internal logic, and enter a dormant state. This signal may be asynchronous to the 80188 clock. The 80188 begins fetching instructions approximately 7 clock cycles after $\overline{RES}$ is returned HIGH. $\overline{RES}$ is required to be LOW for greater than 4 clock cycles and is internally synchronized. For proper initialization, the LOW-to-HIGH transition of $\overline{RES}$ must occur no sooner than 50 microseconds after power up. This input is provided with a Schmitt-trigger to facilitate power-on $\overline{RES}$ generation via an RC network. When $\overline{RES}$ occurs, the 80188 will drive the status lines to an inactive level for one clock, and then tri-state them.
$\overline{TEST}$	47	I	**$\overline{TEST}$:** Is examined by the WAIT instruction. If the $\overline{TEST}$ input is HIGH when "WAIT" execution begins, instruction execution will suspend. $\overline{TEST}$ will be resampled until it goes LOW, at which time execution will resume. If interrupts are enabled while the 80188 is waiting for $\overline{TEST}$, interrupts will be serviced. This input is synchronized internally.
TMR IN 0, TMR IN 1	20 21	I I	**TIMER INPUTS:** Are used either as clock or control signals, depending upon the programmed timer mode. These inputs are active HIGH (or LOW-to-HIGH transitions are counted) and internally synchronized.
TMR OUT 0, TMR OUT 1	22 23	O O	**TIMER OUTPUTS:** Are used to provide single pulse or continuous waveform generation, depending upon the timer mode selected.
DRQ0, DRQ1	18 19	I I	**DMA REQUEST:** Is driven HIGH by an external device when it desires that a DMA channel (Channel 0 or 1) perform a transfer. These signals are active HIGH, level-triggered, and internally synchronized.
NMI	46	I	**NON-MASKABLE INTERRUPT:** Is an edge-triggered input which causes a type 2 interrupt. NMI is not maskable internally. A transition from a LOW to HIGH initiates the interrupt at the next instruction boundary. NMI is latched internally. An NMI duration of one clock or more will guarantee service. This input is internally synchronized.

Table 1. 80188 Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
INT0, INT1, INT2/$\overline{\text{INTA0}}$, INT3/$\overline{\text{INTA1}}$	45, 44 42 41	I I/O I/O	**MASKABLE INTERRUPT REQUESTS:** Can be requested by strobing one of these pins. When configured as inputs, these pins are active HIGH. Interrupt Requests are synchronized internally. INT2 and INT3 may be configured via software to provide active-LOW interrupt-acknowledge output signals. All interrupt inputs may be configured via software to be either edge- or level-triggered. To ensure recognition, all interrupt requests must remain active until the interrupt is acknowledged. When iRMX mode is selected, the function of these pins changes (see Interrupt Controller section of this data sheet).
A19/S6, A18/S5, A17/S4, A16/S3	65 66 67 68	O O O O	**ADDRESS BUS OUTPUTS (16–19) and BUS CYCLE STATUS (3–6):** Reflect the four most significant address bits during T_1. These signals are active HIGH. During T_2, T_3, T_W, and T_4, status information is available on these lines as encoded below: **Low** / **High** S6: Processor Cycle / DMA Cycle S3, S4, and S5 are defined as LOW during T_2–T_4.
AD7–AD0	2, 4, 6, 8 11, 13, 15, 17	I/O	**ADDRESS/DATA BUS (0-7):** Signals constitute the time multiplexed memory or I/O address (T_1) and data (T_2, T_3, T_W, and T_4) bus. The bus is active HIGH.
A15–A8	1, 3, 5, 7 10, 12, 14, 16	O	**ADDRESS-ONLY BUS (8-15):** Containing valid address from T_1-T_4. The bus is active HIGH.
S7	64	O	This signal is always HIGH to indicate that the 80188 has an 8-bit data bus, and is tri-state OFF during bus HOLD.
ALE/QS0	61	O	**ADDRESS LATCH ENABLE/QUEUE STATUS 0:** Is provided by the 80188 to latch the address into the 8282/8283 address latches. ALE is active HIGH. Addresses are guaranteed to be valid on the trailing edge of ALE. The ALE rising edge is generated off the rising edge of the CLKOUT immediately preceding T_1 of the associated bus cycle, effectively one-half clock cycle earlier than in the standard 8088. The trailing edge is generated off the CLKOUT rising edge in T_1 as in the 8088. Note that ALE is never floated.
$\overline{\text{WR}}$/QS1	63	O	**WRITE STROBE/QUEUE STATUS 1:** Indicates that the data on the bus is to be written into a memory or an I/O device. $\overline{\text{WR}}$ is active for T_2, T_3, and T_W of any write cycle. It is active LOW, and floats during "HOLD." It is driven HIGH for one clock during Reset, and then floated. When the 80188 is in queue status mode, the ALE/QS0 and $\overline{\text{WR}}$/QS1 pins provide information about processor/instruction queue interaction. **QS1** / **QS0** / **Queue Operation** 0 / 0 / No Queue Operation 0 / 1 / First Opcode Byte Fetched from the Queue 1 / 1 / Subsequent Byte Fetched from the Queue 1 / 0 / Empty the Queue

Table 1. 80188 Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
$\overline{RD}/\overline{QSMD}$	62	O	**READ STROBE:** Indicates that the 80188 is performing a memory or I/O read cycle. $\overline{RD}$ is active LOW for T_2, T_3, and T_W of any read cycle. It is guaranteed not to go LOW in T_2 until after the Address Bus is floated. $\overline{RD}$ is active LOW, and floats during "HOLD". $\overline{RD}$ is driven HIGH for one clock during Reset, and then the output driver is floated. A weak internal pull-up mechanism on the $\overline{RD}$ line holds it HIGH when the line is not driven. During RESET the pin is sampled to determine whether the 80188 should provide ALE, $\overline{WR}$, and $\overline{RD}$, or if the Queue-Status should be provided. $\overline{RD}$ should be connected to GND to provide Queue-Status data.
ARDY	55	I	**ASYNCHRONOUS READY:** Informs the 80188 that the addressed memory space or I/O device will complete a data transfer. The ARDY input pin will accept an asynchronous input, and is active HIGH. Only the rising edge is internally synchronized by the 80188. This means that the falling edge of ARDY must be synchronized to the 80188 clock. If connected to V_{CC}, no WAIT states are inserted. Asynchronous ready (ARDY) or synchronous ready (SRDY) must be active to terminate a bus cycle. If unused, this line should be tied LOW.
SRDY	49	I	**SYNCHRONOUS READY:** Must be synchronized externally to the 80188. The use of SRDY provides a relaxed system-timing specification on the Ready input. This is accomplished by eliminating the one-half clock cycle which is required for internally resolving the signal level when using the ARDY input. This line is active HIGH. If this line is connected to V_{CC}, no WAIT states are inserted. Asynchronous ready (ARDY) or synchronous ready (SRDY) must be active before a bus cycle is terminated. If unused, this line should be tied LOW.
$\overline{LOCK}$	48	O	**$\overline{LOCK}$:** Output indicates that other system bus masters are not to gain control of the system bus while $\overline{LOCK}$ is active LOW. The $\overline{LOCK}$ signal is requested by the LOCK prefix instruction and is activated at the beginning of the first data cycle associated with the instruction following the LOCK prefix. It remains active until the completion of the instruction following the LOCK prefix. No prefetches will occur while $\overline{LOCK}$ is asserted. When executing more than one LOCK instruction, always make sure there are 6 bytes of code between the end of the first LOCK instruction and the start of the second LOCK instruction. $\overline{LOCK}$ is active LOW, is driven HIGH for one clock during RESET, and then floated.
$\overline{S0}$, $\overline{S1}$, $\overline{S2}$	52–54	O	**BUS CYCLE STATUS $\overline{S0}$–$\overline{S2}$:** Are encoded to provide bus-transaction information:

80188 Bus Cycle Status Information

$\overline{S2}$	$\overline{S1}$	$\overline{S0}$	Bus Cycle Initiated
0	0	0	Interrupt Acknowledge
0	0	1	Read I/O
0	1	0	Write I/O
0	1	1	Halt
1	0	0	Instruction Fetch
1	0	1	Read Data from Memory
1	1	0	Write Data to Memory
1	1	1	Passive (no bus cycle)

The status pins float during "HOLD."
$\overline{S2}$ may be used as a logical M/$\overline{IO}$ indicator, and $\overline{S1}$ as a DT/$\overline{R}$ indicator.
The status lines are driven HIGH for one clock during Reset, and then floated until a bus cycle begins.

Table 1. 80188 Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
HOLD (input) HLDA (output)	50 51	I O	**HOLD:** Indicates that another bus master is requesting the local bus. The HOLD input is active HIGH. HOLD may be asynchronous with respect to the 80188 clock. The 80188 will issue a HLDA in response to a HOLD request at the end of T_4 or T_I. Simultaneous with the issuance of HLDA, the 80188 will float the local bus and control lines. After HOLD is detected as being LOW, the 80188 will lower HLDA. When the 80188 needs to run another bus cycle, it will again drive the local bus and control lines.
$\overline{UCS}$	34	O	**UPPER MEMORY CHIP SELECT:** Is an active LOW output whenever a memory reference is made to the defined upper portion (1K–256K block) of memory. This line is not floated during bus HOLD. The address range activating $\overline{UCS}$ is software programmable.
$\overline{LCS}$	33	O	**LOWER MEMORY CHIP SELECT:** Is active LOW whenever a memory reference is made to the defined lower portion (1K–256K) of memory. This line is not floated during bus HOLD. The address range activating $\overline{LCS}$ is software programmable.
$\overline{MCS0}$–3	38, 37, 36, 35	O	**MID-RANGE MEMORY CHIP SELECT SIGNALS:** Are active LOW when a memory reference is made to the defined mid-range portion of memory (8K–512K). These lines are not floated during bus HOLD. The address ranges activating $\overline{MCS0}$–3 are software programmable.
$\overline{PCS0}$–4	25, 27-30	O	**PERIPHERAL CHIP SELECT SIGNALS 0–4:** Are active LOW when a reference is made to the defined peripheral area (64K byte I/O space). These lines are not floated during bus HOLD. The address ranges activating $\overline{PCS0}$–4 are software programmable.
$\overline{PCS5}$/A1	31	O	**PERIPHERAL CHIP SELECT 5 or LATCHED A1:** May be programmed to provide a sixth peripheral chip select, or to provide an internally latched A1 signal. The address range activating $\overline{PCS5}$ is software programmable. When programmed to provide latched A1, rather than $\overline{PCS5}$, this pin will retain the previously latched value of A1 during a bus HOLD. A1 is active HIGH.
$\overline{PCS6}$/A2	32	O	**PERIPHERAL CHIP SELECT 6 or LATCHED A2:** May be programmed to provide a seventh peripheral chip select, or to provide an internally latched A2 signal. The address range activating $\overline{PCS6}$ is software programmable. When programmed to provide latched A2, rather than $\overline{PCS6}$, this pin will retain the previously latched value of A2 during a bus HOLD. A2 is active HIGH.
DT/$\overline{R}$	40	O	**DATA TRANSMIT/RECEIVE:** Controls the direction of data flow through the external 8286/8287 data bus transceiver. When LOW, data is transferred to the 80188. When HIGH the 80188 places write data on the data bus.
$\overline{DEN}$	39	O	**DATA ENABLE:** Is provided as an 8286/8287 data bus transceiver output enable. $\overline{DEN}$ is active LOW during each memory and I/O access. $\overline{DEN}$ is HIGH whenever DT/$\overline{R}$ changes state.

FUNCTIONAL DESCRIPTION

Introduction

The following Functional Description describes the base architecture of the 80188. This architecture is common to the 8086, 8088 and 80286 microprocessor families as well. The 80188 is a very high integration 8-bit microprocessor. It combines 15–20 of the most common microprocessor system components onto one chip while providing twice the performance of the standard 8088. The 80188 is object code compatible with the 8086, 8088 microprocessors and adds 10 new instruction types to the existing 8086, 8088 instruction set.

80188 BASE ARCHITECTURE

The 8086, 8088, 80186, 80188 and 80286 family all contain the same basic set of registers, instructions, and addressing modes. The 80188 processor is upward compatible with the 8086, 8088, 80186, and 80286 CPUs.

Register Set

The 80188 base architecture has fourteen registers as shown in Figures 3a and 3b. These registers are grouped into the following categories.

GENERAL REGISTERS

Eight 16-bit general purpose registers may be used to contain arithmetic and logical operands. Four of these (AX, BX, CX, and DX) can be used as 16-bit registers or split into pairs of separate 8-bit registers.

SEGMENT REGISTERS

Four 16-bit special purpose registers select, at any given time, the segments of memory that are immediately addressable for code, stack, and data. (For usage, refer to Memory Organization.)

BASE AND INDEX REGISTERS

Four of the general purpose registers may also be used to determine offset addresses of operands in memory. These registers may contain base addresses or indexes to particular locations within a segment. The addressing mode selects the specific registers for operand and address calculations.

STATUS AND CONTROL REGISTERS

Two 16-bit special purpose registers record or alter certain aspects of the 80188 processor state. These are the Instruction Pointer Register, which contains the offset address of the next sequential instruction to be executed, and the Status Word Register, which contains status and control flag bits (see Figures 3a and 3b).

STATUS WORD DESCRIPTION

The Status Word records specific characteristics of the result of logical and arithmetic instructions (bits 0, 2, 4, 6, 7, and 11) and controls the operation of the 80188 within a given operating mode (bits 8, 9, and 10). The Status Word Register is 16-bits wide. The function of the Status Word bits is shown in Table 2.

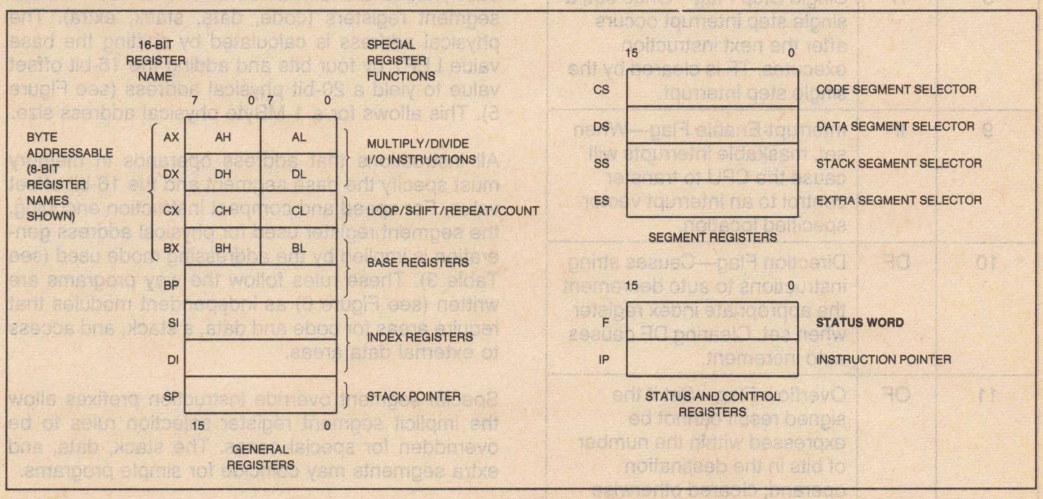


Figure 3a. 80188 General Purpose Register Set

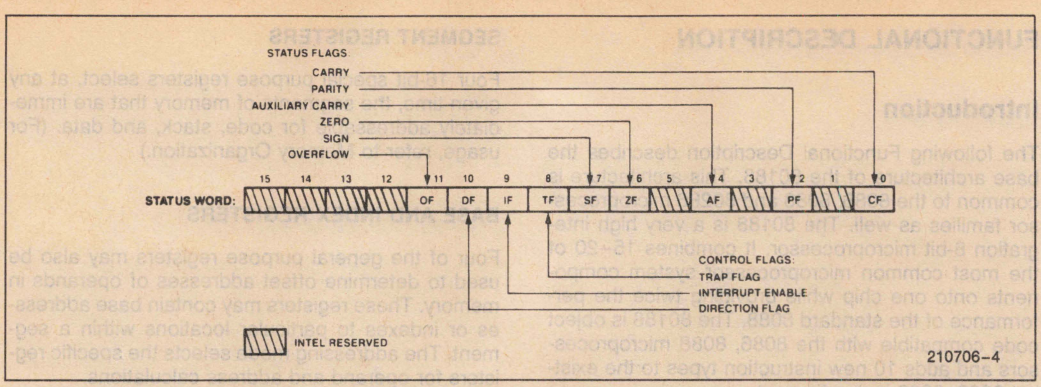


Figure 3b. Status Word Format

Table 2. Status Word Bit Functions

Bit Position	Name	Function
0	CF	Carry Flag—Set on high-order bit carry or borrow; cleared otherwise
2	PF	Parity Flag—Set if low-order 8 bits of result contain an even number of 1-bits; cleared otherwise
4	AF	Set on carry from or borrow to the low order four bits of AL; cleared otherwise
6	ZF	Zero Flag—Set if result is zero; cleared otherwise
7	SF	Sign Flag—Set equal to high-order bit of result (0 if positive, 1 if negative)
8	TF	Single Step Flag—Once set, a single step interrupt occurs after the next instruction executes. TF is cleared by the single step interrupt.
9	IF	Interrupt-Enable Flag—When set, maskable interrupts will cause the CPU to transfer control to an interrupt vector specified location.
10	DF	Direction Flag—Causes string instructions to auto decrement the appropriate index register when set. Clearing DF causes auto increment.
11	OF	Overflow Flag—Set if the signed result cannot be expressed within the number of bits in the destination operand; cleared otherwise

Instruction Set

The instruction set is divided into seven categories: data transfer, arithmetic, shift/rotate/logical, string manipulation, control transfer, high-level instructions, and processor control. These categories are summarized in Figure 4.

An 80188 instruction can reference anywhere from zero to several operands. An operand can reside in a register, in the instruction itself, or in memory. Specific operand addressing modes are discussed later in this data sheet.

Memory Organization

Memory is organized in sets of segments. Each segment is a linear contiguous sequence of up to 64K (2^{16}) 8-bit bytes. Memory is addressed using a two-component address (a pointer) that consists of a 16-bit base segment and a 16-bit offset. The 16-bit base values are contained in one of four internal segment registers (code, data, stack, extra). The physical address is calculated by shifting the base value LEFT by four bits and adding the 16-bit offset value to yield a 20-bit physical address (see Figure 5). This allows for a 1 MByte physical address size.

All instructions that address operands in memory must specify the base segment and the 16-bit offset value. For speed and compact instruction encoding, the segment register used for physical address generation is implied by the addressing mode used (see Table 3). These rules follow the way programs are written (see Figure 6) as independent modules that require areas for code and data, a stack, and access to external data areas.

Special segment override instruction prefixes allow the implicit segment register selection rules to be overridden for special cases. The stack, data, and extra segments may coincide for simple programs.

GENERAL PURPOSE	
MOV	Move byte or word
PUSH	Push word onto stack
POP	Pop word off stack
PUSHA	Push all registers on stack
POPA	Pop all registers from stack
XCHG	Exchange byte or word
XLAT	Translate byte
INPUT/OUTPUT	
IN	Input byte or word
OUT	Output byte or word
ADDRESS OBJECT	
LEA	Load effective address
LDS	Load pointer using DS
LES	Load pointer using ES
FLAG TRANSFER	
LAHF	Load AH register from flags
SAHF	Store AH register in flags
PUSHF	Push flags onto stack
POPF	Pop flags off stack
ADDITION	
ADD	Add byte or word
ADC	Add byte or word with carry
INC	Increment byte or word by 1
AAA	ASCII adjust for addition
DAA	Decimal adjust for addition
SUBTRACTION	
SUB	Subtract byte or word
SBB	Subtract byte or word with borrow
DEC	Decrement byte or word by 1
NEG	Negate byte or word
CMP	Compare byte or word
AAS	ASCII adjust for subtraction
DAS	Decimal adjust for subtraction
MULTIPLICATION	
MUL	Multiply byte or word unsigned
IMUL	Integer multiply byte or word
AAM	ASCII adjust for multiply
DIVISION	
DIV	Divide byte or word unsigned
IDIV	Integer divide byte or word
AAD	ASCII adjust for division
CBW	Convert byte to word
CWD	Convert word to doubleword

MOVS	Move byte or word string
INS	Input bytes or word string
OUTS	Output bytes or word string
CMPS	Compare byte or word string
SCAS	Scan byte or word string
LODS	Load byte or word string
STOS	Store byte or word string
REP	Repeat
REPE/REPZ	Repeat while equal/zero
REPNE/REPNZ	Repeat while not equal/not zero
LOGICALS	
NOT	"Not" byte or word
AND	"And" byte or word
OR	"Inclusive or" byte or word
XOR	"Exclusive or" byte or word
TEST	"Test" byte or word
SHIFTS	
SHL/SAL	Shift logical/arithmetic left byte or word
SHR	Shift logical right byte or word
SAR	Shift arithmetic right byte or word
ROTATES	
ROL	Rotate left byte or word
ROR	Rotate right byte or word
RCL	Rotate through carry left byte or word
RCR	Rotate through carry right byte or word
FLAG OPERATIONS	
STC	Set carry flag
CLC	Clear carry flag
CMC	Complement carry flag
STD	Set direction flag
CLD	Clear direction flag
STI	Set interrupt enable flag
CLI	Clear interrupt enable flag
EXTERNAL SYNCHRONIZATION	
HLT	Halt until interrupt or reset
WAIT	Wait for $\overline{\text{TEST}}$ pin active
ESC	Escape to extension processor
LOCK	Lock bus during next instruction
NO OPERATION	
NOP	No operation
HIGH LEVEL INSTRUCTIONS	
ENTER	Format stack for procedure entry
LEAVE	Restore stack for procedure exit
BOUND	Detects values outside prescribed range

Figure 4. 80188 Instruction Set

CONDITIONAL TRANSFERS	
JA/JNBE	Jump if above/not below nor equal
JAE/JNB	Jump if above or equal/not below
JB/JNAE	Jump if below/not above nor equal
JBE/JNA	Jump if below or equal/not above
JC	Jump if carry
JE/JZ	Jump if equal/zero
JG/JNLE	Jump if greater/not less nor equal
JGE/JNL	Jump if greater or equal/not less
JL/JNGE	Jump if less/not greater nor equal
JLE/JNG	Jump if less or equal/not greater
JNC	Jump if not carry
JNE/JNZ	Jump if not equal/not zero
JNO	Jump if not overflow
JNP/JPO	Jump if not parity/parity odd
JNS	Jump if not sign
JO	Jump if overflow
JP/JPE	Jump if parity/parity even
JS	Jump if sign
UNCONDITIONAL TRANSFERS	
CALL	Call procedure
RET	Return from procedure
JMP	Jump
ITERATION CONTROLS	
LOOP	Loop
LOOPE/LOOPZ	Loop if equal/zero
LOOPNE/LOOPNZ	Loop if not equal/not zero
JCXZ	Jump if register CX = 0
INTERRUPTS	
INT	Interrupt
INTO	Interrupt if overflow
IRET	Interrupt return

Figure 4. 80188 Instruction Set (Continued)

To access operands that do not reside in one of the four immediately available segments, a full 32-bit pointer can be used to reload both the base (segment) and offset values.

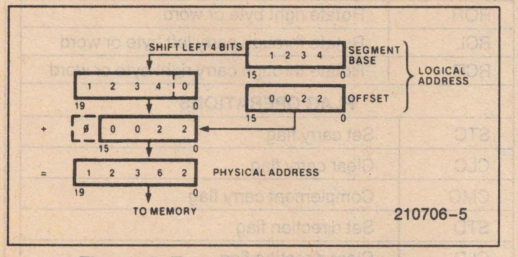


Figure 5. Two Component Address

Table 3. Segment Register Selection Rules

Memory Reference Needed	Segment Register Used	Implicit Segment Selection Rule
Instructions	Code (CS)	Instruction prefetch and immediate data.
Stack	Stack (SS)	All stack pushes and pops; any memory references which use BP Register as a base register.
External Data (Global)	Extra (ES)	All string instruction references which use the DI register as an index.
Local Data	Data (DS)	All other data references.

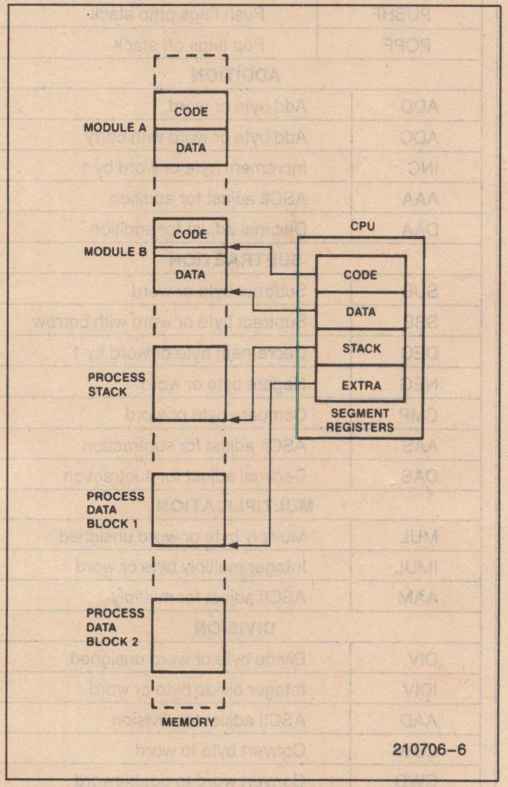


Figure 6. Segmented Memory Helps Structure Software

Addressing Modes

The 80188 provides eight categories of addressing modes to specify operands. Two addressing modes are provided for instructions that operate on register or immediate operands:

- *Register Operand Mode:* The operand is located in one of the 8- or 16-bit general registers.
- *Immediate Operand Mode:* The operand is included in the instruction.

Six modes are provided to specify the location of an operand in a memory segment. A memory operand address consists of two 16-bit components: a segment base and an offset. The segment base is supplied by a 16-bit segment register either implicitly chosen by the addressing mode or explicitly chosen by a segment override prefix. The offset, also called the effective address, is calculated by summing any combination of the following three address elements:

- the *displacement* (an 8- or 16-bit immediate value contained in the instruction);
- the *base* (contents of either the BX or BP base registers); and
- the *index* (contents of either the SI or DI index registers).

Any carry out from the 16-bit addition is ignored. Eight-bit displacements are sign extended to 16-bit values.

Combinations of these three address elements define the six memory addressing modes, described below.

- *Direct Mode:* The operand's offset is contained in the instruction as an 8- or 16-bit displacement element.
- *Register Indirect Mode:* The operand's offset is in one of the registers SI, DI, BX, or BP.
- *Based Mode:* The operand's offset is the sum of an 8- or 16-bit displacement and the contents of a base register (BX or BP).
- *Indexed Mode:* The operand's offset is the sum of an 8- or 16-bit displacement and the contents of an index register (SI or DI).
- *Based Indexed Mode:* The operand's offset is the sum of the contents of a base register and an index register.
- *Based Indexed Mode with Displacement:* The operand's offset is the sum of a base register's contents, an index register's contents, and an 8- or 16-bit displacement.

Data Types

The 80188 directly supports the following data types:

- *Integer:* A signed binary numeric value contained in an 8-bit byte or a 16-bit word. All operations assume a 2's complement representation. Signed 32- and 64-bit integers are supported using the 80188/20 Numeric Data Processor.
- *Ordinal:* An unsigned binary numeric value contained in an 8-bit byte or a 16-bit word.
- *Pointer:* A 16- or 32-bit quantity, composed of a 16-bit offset component or a 16-bit segment base component in addition to a 16-bit offset component.
- *String:* A contiguous sequence of bytes or words. A string may contain from 1 to 64K bytes.
- *ASCII:* A byte representation of alphanumeric and control characters using the ASCII standard of character representation.
- *BCD:* A byte (unpacked) representation of the decimal digits 0–9.
- *Packed BCD:* A byte (packed) representation of two decimal digits (0–9). One digit is stored in each nibble (4-bits) of the byte.
- *Floating Point:* A signed 32-, 64-, or 80-bit real number representation. (Floating point operands are supported using the 80188/20 Numeric Data Processor configuration.)

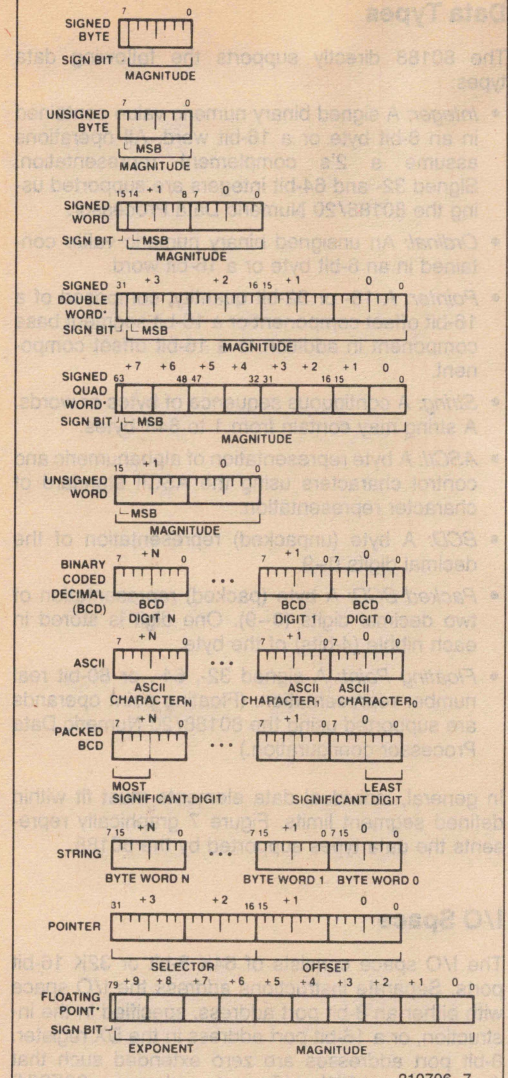
In general, individual data elements must fit within defined segment limits. Figure 7 graphically represents the data types supported by the 80188.

I/O Space

The I/O space consists of 64K 8-bit or 32K 16-bit ports. Separate instructions address the I/O space with either an 8-bit port address, specified in the instruction, or a 16-bit port address in the DX register. 8-bit port addresses are zero extended such that A_{15}–A_8 are LOW. I/O port addresses 00F8(H) through 00FF(H) are reserved.

Interrupts

An interrupt transfers execution to a new program location. The old program address (CS:IP) and machine state (Status Word) are saved on the stack to allow resumption of the interrupted program. Interrupts fall into three classes: hardware initiated, INT instructions, and instruction exceptions. Hardware initiated interrupts occur in response to an external input and are classified as non-maskable or maskable.



NOTE:
*Supported By 80188/20 Numeric Data Processor Configuration.

Figure 7. 80188 Supported Data Types

Programs may cause an interrupt with an INT instruction. Instruction exceptions occur when an unusual condition, which prevents further instruction processing, is detected while attempting to execute an instruction. If the exception was caused by executing an ESC instruction with the ESC trap bit set in the relocation register, the return instruction will point to the ESC instruction, or to the segment override prefix immediately preceding the ESC instruction if the prefix was present. In all other cases, the return address from an exception will point at the instruction immediately following the instruction causing the exception.

A table containing up to 256 pointers defines the proper interrupt service routine for each interrupt. Interrupts 0–31, some of which are used for instruction exceptions, are reserved. Table 4 shows the 80188 predefined types and default priority levels. For each interrupt, an 8-bit vector must be supplied to the 80188 which identifies the appropriate table entry. Exceptions supply the interrupt vector internally. In addition, internal peripherals and noncascaded external interrupts will generate their own vectors through the internal interrupt controller. INT instructions contain or imply the vector and allow access to all 256 interrupts. Maskable hardware initiated interrupts supply the 8-bit vector to the CPU during an interrupt acknowledge bus sequence. Non-maskable hardware interrupts use a predefined internally supplied vector.

Interrupt Sources

The 80188 can service interrupts generated by software or hardware. The software interrupts are generated by specific instructions (INT, ESC, unused OP, etc.) or the results of conditions specified by instructions (array bounds check, INT0, DIV, IDIV, etc.). All interrupt sources are serviced by an indirect call through an element of a vector table. This vector table is indexed by using the interrupt vector type (Table 4), multiplied by four. All hardware-generated interrupts are sampled at the end of each instruction. Thus, the software interrupts will begin service first. Once the service routine is entered and interrupts are enabled, any hardware source of sufficient priority can interrupt the service routine in progress.

The software generated 80188 interrupts are described below.

DIVIDE ERROR EXCEPTION (TYPE 0)

Generated when a DIV or IDIV instruction quotient cannot be expressed in the number of bits in the destination.

SINGLE-STEP INTERRUPT (TYPE 1)

Generated after most instructions if the TF flag is set. Interrupts will not be generated after prefix instructions (e.g., REP), instructions which modify segment registers (e.g., POP DS), or the WAIT instruction.

NON-MASKABLE INTERRUPT—NMI (TYPE 2)

An external interrupt source which cannot be masked.

Table 4. 80188 Interrupt Vectors

Interrupt Name	Vector Type	Default Priority	Related Instructions
Divide Error Exception	0	*1	DIV, IDIV
Single Step Interrupt	1	12**2	All
NMI	2	1	All
Breakpoint Interrupt	3	*1	INT
INT0 Detected Overflow Exception	4	*1	INT0
Array Bounds Exception	5	*1	BOUND
Unused-Opcode Exception	6	*1	Undefined Opcodes
ESC Opcode Exception	7	*1***	ESC Opcodes
Timer 0 Interrupt	8	2A****	
Timer 1 Interrupt	18	2B****	
Timer 2 Interrupt	19	2C****	
Reserved	9	3	
DMA 0 Interrupt	10	4	
DMA 1 Interrupt	11	5	
INT0 Interrupt	12	6	
INT1 Interrupt	13	7	
INT2 Interrupt	14	8	
INT3 Interrupt	15	9	

NOTES:
*1. These are generated as the result of an instruction execution.
**2. This is handled as in the 8088.
****3. All three timers constitute one source of request to the interrupt controller. The Timer Interrupts all have the same default priority level with respect to all other interrupt sources. However, they have a defined priority ordering amongst themselves. (Priority 2A is higher priority than 2B.) Each Timer Interrupt has a separate vector type number.
4. Default priorities for the interrupt sources are used only if the user does not program each source into a unique priority level.
***5. An escape opcode will cause a trap only if the proper bit is set in the peripheral control block relocation register.

BREAKPOINT INTERRUPT (TYPE 3)

A one-byte version of the INT instruction. It uses 12 as an index into the service routine address table (because it is a type 3 interrupt).

INT0 DETECTED OVERFLOW EXCEPTION (TYPE 4)

Generated during an INT0 instruction if the 0F bit is set.

ARRAY BOUNDS EXCEPTION (TYPE 5)

Generated during a BOUND instruction if the array index is outside the array bounds. The array bounds are located in memory at a location indicated by one of the instruction operands. The other operand indicates the value of the index to be checked.

UNUSED OPCODE EXCEPTION (TYPE 6)

Generated if execution is attempted on undefined opcodes.

ESCAPE OPCODE EXCEPTION (TYPE 7)

Generated if execution is attempted of ESC opcodes (D8H–DFH). This exception will only be generated if a bit in the relocation register is set. The return address of this exception will point to the ESC instruction causing the exception. If a segment override prefix preceded the ESC instruction, the return address will point to the segment override prefix.

Hardware-generated interrupts are divided into two groups: maskable interrupts and non-maskable interrupts. The 80188 provides maskable hardware interrupt request pins INT0–INT3. In addition, maskable interrupts may be generated by the 80188 integrated DMA controller and the integrated timer unit. The vector types for these interrupts are shown in Table 4. Software enables these inputs by setting the Interrupt Flag bit (IF) in the Status Word. The interrupt controller is discussed in the peripheral section of this data sheet.

Further maskable interrupts are disabled while servicing an interrupt because the IF bit is reset as part of the response to an interrupt or exception. The saved Status Word will reflect the enable status of the processor prior to the interrupt. The interrupt flag will remain zero unless specifically set. The interrupt return instruction restores the Status Word, thereby restoring the original status of IF bit. If the interrupt return re-enables interrupts, and another interrupt is pending, the 80188 will immediately service the highest-priority interrupt pending, i.e., no instructions of the main line program will be executed.

Non-Maskable Interrupt Request (NMI)

A non-maskable interrupt (NMI) is also provided. This interrupt is serviced regardless of the state of the IF bit. A typical use of NMI would be to activate a power failure routine. The activation of this input causes an interrupt with an internally supplied vector value of 2. No external interrupt acknowledge sequence is performed. The IF bit is cleared at the beginning of an NMI interrupt to prevent maskable interrupts from being serviced.

Single-Step Interrupt

The 80188 has an internal interrupt that allows programs to execute one instruction at a time. It is called the single-step interrupt and is controlled by the single-step flag bit (TF) in the Status Word. Once this bit is set, an internal single-step interrupt will occur after the next instruction has been executed. The interrupt clears the TF bit and uses an internally supplied vector of 1. The IRET instruction is used to set the TF bit and transfer control to the next instruction to be single-stepped.

Initialization and Processor Reset

Processor initialization or startup is accomplished by driving the $\overline{RES}$ input pin LOW. $\overline{RES}$ forces the 80188 to terminate all execution and local bus activity. No instruction or bus activity will occur as long as $\overline{RES}$ is active. After $\overline{RES}$ becomes inactive and an internal processing interval elapses, the 80188 begins execution with the instruction at physical location FFFF0(H). $\overline{RES}$ also sets some registers to predefined values as shown in Table 5.

Table 5. 80188 Initial Register State after RESET

Status Word	F002(H)
Instruction Pointer	0000(H)
Code Segment	FFFF(H)
Data Segment	0000(H)
Extra Segment	0000(H)
Stack Segment	0000(H)
Relocation Register	20FF(H)
UMCS	FFFB(H)

THE 80188 COMPARED TO THE 80186

The 80188 CPU is an 8-bit processor designed around the 80186 internal structure. Most internal functions of the 80188 are identical to the equivalent 80186 functions. The 80188 handles the external bus the same way the 80186 does with the distinction of handling only 8 bits at a time. Sixteen bit operands are fetched or written in two consecutive bus cycles. Both processors will appear identical to the software engineer, with the exception of execution time. The internal register structure is identical and all instructions have the same end result. The differences between the 80188 and the 80186 are outlined below. Internally, there are three differences between the 80188 and the 80186. All changes are related to the 8-bit bus interface.

- The queue length is 4 bytes in the 80188, whereas the 80186 queue contains 6 bytes, or three words. The queue was shortened to prevent overuse of the bus by the BIU when prefetching instructions. This was required because of the additional time necessary to fetch instructions 8 bits at a time.
- To further optimize the queue, the prefetching algorithm was changed. The 80188 BIU will fetch a new instruction to load into the queue each time there is a 1-byte hole (space available) in the queue. The 80186 waits until a 2-byte space is available.
- The internal execution time of the instruction is affected by the 8-bit interface. All 16-bit fetches and writes from/to memory take an additional four clock cycles. The CPU may also be limited by the speed of instruction fetches when a series of simple operations occur. When the more sophisticated instructions of the 80188 are being used, the queue has time to fill and the execution proceeds as fast as the execution unit will allow.

The 80188 and 80186 are completely software compatible by virtue of their identical execution units. Software that is system dependent may not be completely transferable, but software that is not system dependent will operate equally well on an 80188 or an 80186.

The hardware interface of the 80188 contains the major differences between the two CPUs. The pin assignments are nearly identical, however, with the following functional changes.

- A8–A15—These pins are only address outputs on the 80188. These address lines are latched internally and remain valid throughout a bus cycle in a manner similar to the 8085 upper address lines.
- $\overline{BHE}$ has no meaning on the 80188 and has been eliminated.

80188 Clock Generator

The 80188 provides an on-chip clock generator for both internal and external clock generation. The clock generator features a crystal oscillator, a divide-by-two counter, synchronous and asynchronous ready inputs, and reset circuitry.

Oscillator

The oscillator circuit of the 80188 is designed to be used with a parallel resonant fundamental mode crystal. This is used as the time base for the 80188. The crystal frequency selected will be double the CPU clock frequency. Use of an LC or RC circuit is not recommended with this oscillator. If an external oscillator is used, it can be connected directly to input pin X1 in lieu of a crystal. The output of the oscillator is not directly available outside the 80188. The recommended crystal configuration is shown in Figure 8.

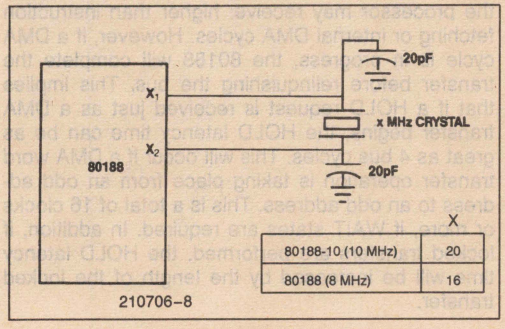


Figure 8. Recommended 80188 Crystal Configuration

The following parameters may be used for choosing a crystal:

Temperature Range:	0 to 70°C
ESR (Equivalent Series Resistance):	30Ω max
C_0 (Shunt Capacitance of Crystal):	7.0 pf max
C_L (Load Capacitance):	20 pf ±2 pf
Drive Level:	1 mW max

Clock Generator

The 80188 clock generator provides the 50% duty cycle processor clock for the 80188. It does this by dividing the oscillator output by 2 forming the symmetrical clock. If an external oscillator is used, the state of the clock generator will change on the falling edge of the oscillator signal. The CLKOUT pin provides the processor clock signal for use outside the 80188. This may be used to drive other system components. All timings are referenced to the output clock.

READY Synchronization

The 80188 provides both synchronous and asynchronous ready inputs. Asynchronous ready synchronization is accomplished by circuitry which samples ARDY in the middle of T_2, T_3 and again in the middle of each T_W until ARDY is sampled HIGH. One-half CLKOUT cycle of resolution time is used. Full synchronization is performed only on the rising edge of ARDY, i.e., the falling edge of ARDY must be synchronized to the CLKOUT signal if it will occur during T_2, T_3, or T_W. HIGH-to-LOW transitions of ARDY must be performed synchronously to the CPU clock.

A second ready input (SRDY) is provided to interface with externally synchronized ready signals. This input is sampled at the end of T_2, T_3 and again at the end of each T_W until it is sampled HIGH. By using this input rather than the asynchronous ready input, the half-clock cycle resolution time penalty is eliminated.

This input must satisfy set-up and hold times to guarantee proper operation of the circuit.

In addition, the 80188, as part of the integrated chip-select logic, has the capability to program WAIT states for memory and peripheral blocks. This is discussed in the Chip Select/Ready Logic description.

RESET Logic

The 80188 provides both a $\overline{RES}$ input pin and a synchronized RESET pin for use with other system components. The $\overline{RES}$ input pin on the 80188 is provided with hysteresis in order to facilitate power-on Reset generation via an RC network. RESET is guaranteed to remain active for at least five clocks given a $\overline{RES}$ input of at least six clocks. RESET may be delayed up to two and one-half clocks behind $\overline{RES}$.

Multiple 80188 processors may be synchronized through the $\overline{RES}$ input pin, since this input resets both the processor and divide-by-two internal counter in the clock generator. In order to insure that the divide-by-two counters all begin counting at the same time, the active going edge of $\overline{RES}$ must satisfy a 25 ns setup time before the falling edge of the

80188 clock input. In addition, in order to insure that all CPUs begin executing in the same clock cycle, the reset must satisfy a 25 ns setup time before the rising edge of the CLKOUT signal of all the processors.

LOCAL BUS CONTROLLER

The 80188 provides a local bus controller to generate the local bus control signals. In addition, it employs a HOLD/HLDA protocol for relinquishing the local bus to other bus masters. It also provides control lines that can be used to enable external buffers and to direct the flow of data on and off the local bus.

Memory/Peripheral Control

The 80188 provides ALE, $\overline{RD}$, and $\overline{WR}$ bus control signals. The $\overline{RD}$ and $\overline{WR}$ signals are used to strobe data from memory to the 80188 or to strobe data from the 80188 to memory. The ALE line provides a strobe to address latches for the multiplexed address/data bus. The 80188 local bus controller does not provide a memory/$\overline{I/O}$ signal. If this is required, the user will have to use the $\overline{S2}$ signal (which will require external latching), make the memory and I/O spaces nonoverlapping, or use only the integrated chip-select circuitry.

Transceiver Control

The 80188 generates two control signals to be connected to 8286/8287 transceiver chips. This capability allows the addition of transceivers for extra buffering without adding external logic. These control lines, DT/$\overline{R}$ and $\overline{DEN}$, are generated to control the flow of data through the transceivers. The operation of these signals is shown in Table 6.

Table 6. Transceiver Control Signals Description

Pin Name	Function
$\overline{DEN}$ (Data Enable)	Enables the output drivers of the transceivers. It is active LOW during memory, I/O, or INTA cycles.
DT/$\overline{R}$ (Data Transmit/ Receive)	Determines the direction of travel through the transceivers. A HIGH level directs data away from the processor during write operations, while a LOW level directs data toward the processor during a read operation.

Local Bus Arbitration

The 80188 uses a HOLD/HLDA system of local bus exchange. This provides an asynchronous bus exchange mechanism. This means multiple masters utilizing the same bus can operate at separate clock frequencies. The 80188 provides a single HOLD/HLDA pair through which all other bus masters may gain control of the local bus. This requires external circuitry to arbitrate which external device will gain control of the bus from the 80188 when there is more than one alternate local bus master. When the 80188 relinquishes control of the local bus, it floats $\overline{DEN}$, $\overline{RD}$, $\overline{WR}$, $\overline{S0}$–$\overline{S2}$, $\overline{LOCK}$, AD0–AD15, A16–A19, $\overline{S7}$, and DT/$\overline{R}$ to allow another master to drive these lines directly.

The 80188 HOLD latency time, i.e., the time between HOLD request and HOLD acknowledge, is a function of the activity occurring in the processor when the HOLD request is received. A HOLD request is the highest-priority activity request which the processor may receive: higher than instruction fetching or internal DMA cycles. However, if a DMA cycle is in progress, the 80188 will complete the transfer before relinquishing the bus. This implies that if a HOLD request is received just as a DMA transfer begins, the HOLD latency time can be as great as 4 bus cycles. This will occur if a DMA word transfer operation is taking place from an odd address to an odd address. This is a total of 16 clocks or more, if WAIT states are required. In addition, if locked transfers are performed, the HOLD latency time will be increased by the length of the locked transfer.

Local Bus Controller and Reset

Upon receipt of a RESET pulse from the $\overline{RES}$ input, the local bus controller will perform the following actions:

- Drive $\overline{DEN}$, $\overline{RD}$, and $\overline{WR}$ HIGH for one clock cycle, then float.

NOTE:
$\overline{RD}$ is also provided with an internal pull-up device to prevent the processor from inadvertently entering Queue Status mode during reset.

- Drive $\overline{S0}$–$\overline{S2}$ to the passive state (all HIGH) and then float.
- Drive $\overline{LOCK}$ HIGH and then float.
- Float AD0–7, A8–19, S7, DT/$\overline{R}$.
- Drive ALE LOW (ALE is never floated).
- Drive HLDA LOW.

INTERNAL PERIPHERAL INTERFACE

All the 80188 integrated peripherals are controlled via 16-bit registers contained within an internal 256-byte control block. This control block may be mapped into either memory or I/O space. Internal logic will recognize the address and respond to the bus cycle. During bus cycles to internal registers, the bus controller will signal the operation externally (i.e., the $\overline{RD}$, $\overline{WR}$, status, address, data, etc., lines will be driven as in a normal bus cycle), but D_{7-0}, SRDY, and ARDY will be ignored. The base address of the control block must be on an even 256-byte boundary (i.e., the lower 8 bits of the base address are all zeros). All of the defined registers within this control block may be read or written by the 80188 CPU at any time. The location of any register contained within the 256-byte control block is determined by the current base address of the control block.

The control block base address is programmed via a 16-bit relocation register contained within the control block at offset FEH from the base address of the control block (see Figure 9). It provides the upper 12 bits of the base address of the control block. Note that mapping the control register block into an address range corresponding to a chip-select range is not recommended (the chip select circuitry is discussed later in this data sheet. In addition, bit 12 of this register determines whether the control block will be mapped into I/O or memory space. If this bit is 1, the control block will be located in memory space, whereas if the bit is 0, the control block will be located in I/O space. If the control register block is mapped into I/O space, the upper 4 bits of the base address must be programmed as 0 (since I/O addresses are only 16 bits wide).

Whenever mapping the 188 peripheral control block to another location, the programming of the relocation register should be done with a byte write (i.e. OUT DX,AL). Any access to the control block is done 16 bits at a time. Thus, internally, the relocation register will get written with 16 bits of the AX register while externally, the BIU will run only one 8 bit bus cycle. If a word instruction is used (i.e. OUT DX,AX), the relocation register will be written on the first bus cycle. The BIU will then run a second bus cycle which is unnecessary. The address of the second bus cycle will no longer be within the control block (i.e. the control block was moved on the first cycle), and therefore, will require the generation of an external ready signal to complete the cycle. For this reason we recommend byte operations to the relocation register. Byte instructions may also be used for the other registers in the control block and will eliminate half of the bus cycles required if a word operation had been specified. Byte operations are only valid on even addresses though, and are undefined on odd addresses.

In addition to providing relocation information for the control block, the relocation register contains bits which place the interrupt controller into iRMX mode, and cause the CPU to interrupt upon encountering ESC instructions. At RESET, the relocation register is set to 20FFH. This causes the control block to start at FF00H in I/O space. An offset map of the 256-byte control register block is shown in Figure 10.

The integrated 80188 peripherals operate semi-autonomously from the CPU. Access to them for the most part is via software read/write of the control and data locations in the control block. Most of these registers can be both read and written. A few dedicated lines, such as interrupts and DMA request provide real-time communication between the CPU and peripherals as in a more conventional system utilizing discrete peripheral blocks. The overall interaction and function of the peripheral blocks has not substantially changed. The data access from/to the 256-byte internal control block will always be 16-bit and done in one bus cycle. Externally the BIU will still run two bus cycles for each 16-bit operation.

CHIP-SELECT/READY GENERATION LOGIC

The 80188 contains logic which provides programmable chip-select generation for both memories and peripherals. In addition, it can be programmed to provide READY (or WAIT state) generation. It can also provide latched address bits A1 and A2. The chip-select lines are active for all memory and I/O cycles in their programmed areas, whether they be generated by the CPU or by the integrated DMA unit.

Memory Chip Selects

The 80188 provides 6 memory chip select outputs for 3 address areas: upper memory, lower memory, and midrange memory. One each is provided for upper memory and lower memory, while four are provided for midrange memory.

The range for each chip select is user-programmable and can be set to 2K, 4K, 8K, 16K, 32K, 64K, 128K (plus 1K and 256K for upper and lower chip selects). In addition, the beginning or base address

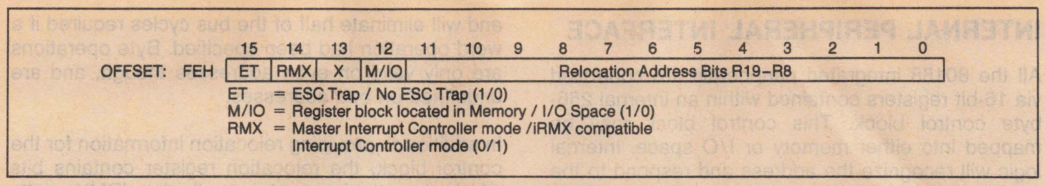


Figure 9. Relocation Register

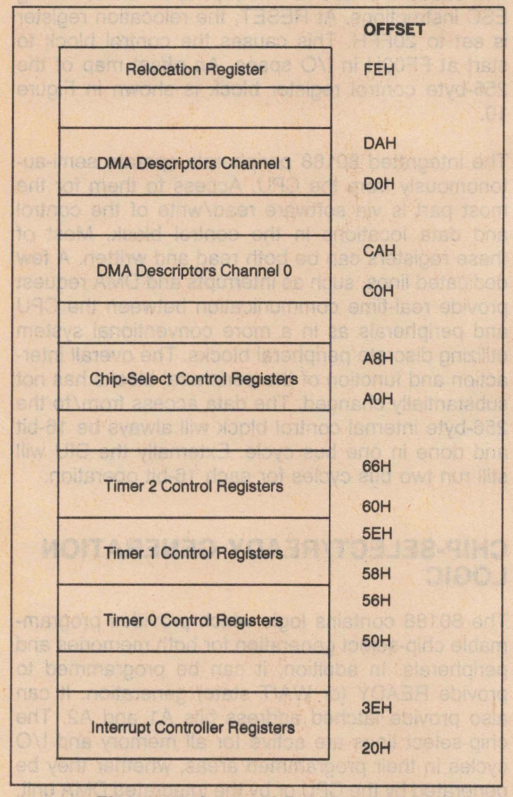


Figure 10. Internal Register Map

of the midrange memory chip select may also be selected. Only one chip select may be programmed to be active for any memory location at a time. All chip select sizes are in bytes.

Upper Memory $\overline{CS}$

The 80188 provides a chip select, called $\overline{UCS}$, for the top of memory. The top of memory is usually used as the system memory because after reset the 80188 begins executing at memory location FFFF0H.

The upper limit of memory defined by this chip select is always FFFFFH, while the lower limit is programmable. By programming the lower limit, the size of the select block is also defined. Table 7 shows the relationship between the base address selected and the size of the memory block obtained.

Table 7. UMCS Programming Values

Starting Address (Base Address)	Memory Block Size	UMCS Value (Assuming R0 = R1 = R2 = 0)
FFC00	1K	FFF8H
FF800	2K	FFB8H
FF000	4K	FF38H
FE000	8K	FE38H
FC000	16K	FC38H
F8000	32K	F838H
F0000	64K	F038H
E0000	128K	E038H
C0000	256K	C038H

The lower limit of this memory block is defined in the UMCS register (see Figure 11). This register is at offset A0H in the internal control block. The legal values for bits 6–13 and the resulting starting address and memory block sizes are given in Table 7. Any combination of bits 6–13 not shown in Table 7 will result in undefined operation. After reset, the UMCS register is programmed for a 1K area. It must be reprogrammed if a larger upper memory area is desired.

Any internally generated 20-bit address whose upper 16 bits are greater than or equal to UMCS (with bits 0–5 "0") will cause UCS to be activated. UMCS bits R2–R0 are used to specify READY mode for the area of memory defined by this chip-select register, as explained below.

Lower Memory $\overline{CS}$

The 80188 provides a chip select for low memory called $\overline{LCS}$. The bottom of memory contains the interrupt vector table, starting at location 00000H.

The lower limit of memory defined by this chip select is always 0H, while the upper limit is programmable. By programming the upper limit, the size of the memory block is also defined. Table 8 shows the relationship between the upper address selected and the size of the memory block obtained.

Table 8. LMCS Programming Values

Upper Address	Memory Block Size	LMCS Value (Assuming R0=R1=R2=0)
003FFH	1K	0038H
007FFH	2K	0078H
00FFFH	4K	00F8H
01FFFH	8K	01F8H
03FFFH	16K	03F8H
07FFFH	32K	07F8H
0FFFFH	64K	0FF8H
1FFFFH	128K	1FF8H
3FFFFH	256K	3FF8H

The upper limit of this memory block is defined in the LMCS register (see Figure 12). This register is at offset A2H in the internal control block. The legal values for bits 6–15 and the resulting upper address and memory block sizes are given in Table 8. Any combination of bits 6–15 not shown in Table 8 will result in undefined operation. After reset, the LMCS register value is undefined. However, the $\overline{LCS}$ chip-select line will not become active until the LMCS register is accessed.

Any internally generated 20-bit address whose upper 16 bits are less than or equal to LMCS (with bits 0–5 "1") will cause $\overline{LCS}$ to be active. LMCS register bits R2–R0 are used to specify the READY mode for the area of memory defined by this chip-select register.

Mid-Range Memory $\overline{CS}$

The 80188 provides four $\overline{MCS}$ lines which are active within a user-locatable memory block. This block can be located anywhere within the 80188 1M byte memory address space exclusive of the areas defined by $\overline{UCS}$ and $\overline{LCS}$. Both the base address and size of this memory block are programmable.

The size of the memory block defined by the mid-range select lines, as shown in Table 9, is determined by bits 8–14 of the MPCS register (see Figure 13). This register is at location A8H in the internal control block. One and only one of bits 8–14 must be set at a time. Unpredictable operation of the $\overline{MCS}$ lines will otherwise occur. Each of the four chip-select lines is active for one of the four equal contiguous divisions of the mid-range block. Thus, if the total block size is 32K, each chip select is active for 8K of memory with $\overline{MCS0}$ being active for the first range and $\overline{MCS3}$ being active for the last range.

The EX and MS in MPCS relate to peripheral functionality as described in a later section.

Table 9. MPCS Programming Values

Total Block Size	Individual Select Size	MPCS Bits 14–8
8K	2K	0000001B
16K	4K	0000010B
32K	8K	0000100B
64K	16K	0001000B
128K	32K	0010000B
256K	64K	0100000B
512K	128K	1000000B

The base address of the mid-range memory block is defined by bits 15–9 of the MMCS register (see Figure 14). This register is at offset A6H in the internal control block. These bits correspond to bits A19–A13 of the 20-bit memory address. Bits A12–A0 of the base address are always 0. The base address may be set at any integer multiple of the size of the total memory block selected. For example, if the mid-range block size is 32K (or the size of the block for which each $\overline{MCS}$ line is active is 8K), the block could be located at 10000H or 18000H, but not at 14000H, since the first few integer multiples of a 32K memory block are 0H, 8000H, 10000H, 18000H, etc. After reset, the contents of both of these registers is undefined. However, none of the $\overline{MCS}$ lines will be active until both the MMCS and MPCS registers are accessed.

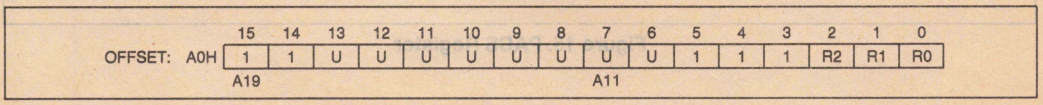


Figure 11. UMCS Register

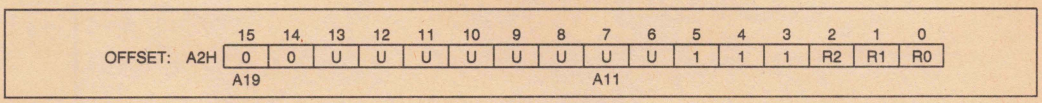


Figure 12. LMCS Register

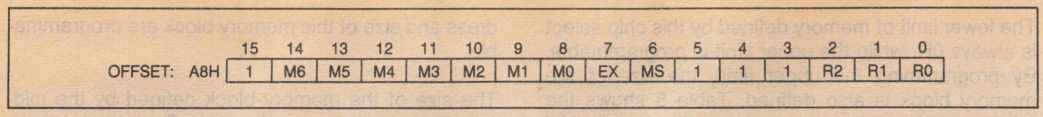


Figure 13. MPCS Register

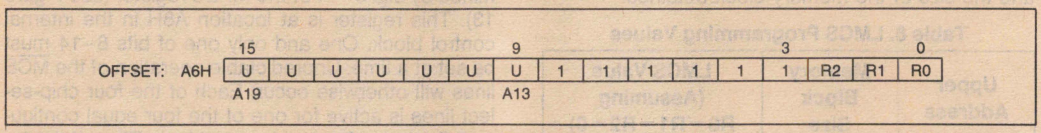


Figure 14. MMCS Register

MMCS bits R2–R0 specify READY mode of operation for all mid-range chip selects. All devices in mid-range memory must use the same number of WAIT states.

The 512K block size for the mid-range memory chip selects is a special case. When using 512K, the base address would have to be at either locations 00000H or 80000H. If it were to be programmed at 00000H when the $\overline{LCS}$ line was programmed, there would be an internal conflict between the $\overline{LCS}$ ready generation logic and the $\overline{MCS}$ ready generation logic. Likewise, if the base address were programmed at 80000H, there would be a conflict with the $\overline{UCS}$ ready generation logic. Since the $\overline{LCS}$ chip-select line does not become active until programmed, while the $\overline{UCS}$ line is active at reset, the memory base can be set only at 00000H. If this base address is selected, however, the $\overline{LCS}$ range must not be programmed.

Peripheral Chip Selects

The 80188 can generate chip selects for up to seven peripheral devices. These chip selects are active for seven contiguous blocks of 128 bytes above a programmable base address. This base address may be located in either memory or I/O space.

Seven $\overline{CS}$ lines called $\overline{PCS0}$–6 are generated by the 80188. The base address is user-programmable; however it can only be a multiple of 1K bytes, i.e., the least significant 10 bits of the starting address are always 0.

$\overline{PCS5}$ and $\overline{PCS6}$ can also be programmed to provide latched address bits A1, A2. If so programmed, they cannot be used as peripheral selects. These outputs can be connected directly to the A0, A1 pins used for selecting internal registers of 8-bit peripheral chips. This scheme simplifies the hardware interface because the 8-bit registers of peripherals are simply treated as 16-bit registers located on even boundaries in I/O space or memory space where only the lower 8-bits of the register are significant: the upper 8-bits are "don't cares."

The starting address of the peripheral chip-select block is defined by the PACS register (see Figure 15). This register is located at offset A4H in the internal control block. Bits 15–6 of this register correspond to bits 19–10 of the 20-bit Programmable Base Address (PBA) of the peripheral chip-select block. Bits 9–0 of the PBA of the peripheral chip-select block are all zeros. If the chip-select block is located in I/O space, bits 12–15 must be programmed zero, since the I/O address is only 16 bits wide. Table 10 shows the address range of each peripheral chip select with respect to the PBA contained in PACS register.

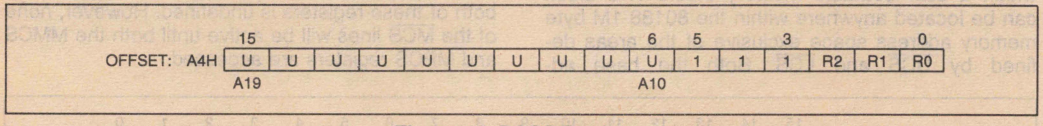


Figure 15. PACS Register

The user should program bits 15–6 to correspond to the desired peripheral base location. PACS bits 0–2 are used to specify READY mode for $\overline{PSC0}$–$\overline{PCS3}$.

Table 10. PCS Address Ranges

PCS Line	Active between Locations
PCS0	PBA —PBA + 127
PCS1	PBA + 128—PBA + 255
PCS2	PBA + 256—PBA + 383
PCS3	PBA + 384—PBA + 511
PCS4	PBA + 512—PBA + 639
PCS5	PBA + 640—PBA + 767
PCS6	PBA + 768—PBA + 895

The mode of operation of the peripheral chip selects is defined by the MPCS register (which is also used to set the size of the mid-range memory chip-select block, see Figure 16). This register is located at offset A8H in the internal control block. Bit 7 is used to select the function of $\overline{PCS5}$ and $\overline{PCS6}$, while bit 6 is used to select whether the peripheral chip selects are mapped into memory or I/O space. Table 11 describes the programming of these bits. After reset, the contents of both the MPCS and the PACS registers are undefined, however none of the PCS lines will be active until both of the MPCS and PACS registers are accessed.

Table 11. MS, EX Programming Values

Bit	Description
MS	1 = Peripherals mapped into memory space.
	0 = Peripherals mapped into I/O space.
EX	0 = 5 $\overline{PCS}$ lines. A1, A2 provided.
	1 = 7 $\overline{PCS}$ lines. A1, A2 are not provided.

MPCS bits 0–2 are used to specify READY mode for $\overline{PCS4}$–$\overline{PCS6}$ as outlined below.

READY Generation Logic

The 80188 can generate a "READY" signal internally for each of the memory or peripheral $\overline{CS}$ lines. The number of WAIT states to be inserted for each peripheral or memory is programmable to provide 0–3 wait states for all accesses to the area for which the chip select is active. In addition, the 80188 may be programmed to either ignore external READY for each chip-select range individually or to factor external READY with the integrated ready generator.

READY control consists of 3 bits for each $\overline{CS}$ line or group of lines generated by the 80188. The interpretation of the ready bits is shown in Table 12.

Table 12. READY Bits Programming

R2	R1	R0	Number of WAIT States Generated
0	0	0	0 wait states, external RDY also used.
0	0	1	1 wait state inserted, external RDY also used.
0	1	0	2 wait states inserted, external RDY also used.
0	1	1	3 wait states inserted, external RDY also used.
1	0	0	0 wait states, external RDY ignored.
1	0	1	1 wait state inserted, external RDY ignored.
1	1	0	2 wait states inserted, external RDY ignored.
1	1	1	3 wait states inserted, external RDY ignored.

The internal ready generator operates in parallel with external READY, not in series if the external READY is used (R2 = 0). This means, for example, if the internal generator is set to insert two wait states, but activity on the external READY lines will insert four wait states, the processor will only insert four wait states, not six. This is because the two wait states generated by the internal generator overlapped the first two wait states generated by the external ready signal. Note that the external ARDY and SRDY lines are always ignored during cycles accessing internal peripherals.

R2–R0 of each control word specifies the READY mode for the corresponding block, with the exception of the peripheral chip selects: R2–R0 of PACS set the $\overline{PCS0}$–3 READY mode, R2–R0 of MPCS set the $\overline{PCS4}$–6 READY mode.

Chip Select/Ready Logic and Reset

Upon reset, the Chip-Select/Ready Logic will perform the following actions:

- All chip-select outputs will be driven HIGH.
- Upon leaving RESET, the $\overline{UCS}$ line will be programmed to provide chip selects to a 1K block with the accompanying READY control bits set at 011 to allow the maximum number of internal wait states in conjunction with external Ready consideration (i.e., UMCS resets to FFFBH).
- No other chip select or READY control registers have any predefined values after RESET. They will not become active until the CPU accesses their control registers. Both the PACS and MPCS registers must be accessed before the $\overline{PCS}$ lines will become active.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OFFSET: A8H	1	M6	M5	M4	M3	M2	M1	M0	EX	MS	1	1	1	R2	R1	R0

Figure 16. MPCS Register

DMA Channels

The 80188 DMA controller provides two independent high-speed DMA channels. Data transfers can occur between memory and I/O spaces (e.g., Memory to I/O) or within the same space (e.g., Memory to Memory or I/O to I/O). Each DMA channel maintains both a 20-bit source and destination pointer which can be optionally incremented or decremented after each data transfer. Each data transfer consumes 2 bus cycles (a minimum of 8 clocks), one cycle to fetch data and the other to store data. This provides a data transfer rate of one MByte/sec at 8 MHz.

DMA Operation

Each channel has six registers in the control block which define each channel's specific operation. The control registers consist of a 20-bit Source pointer (2 words), a 20-bit Destination pointer (2 words), a 16-bit Transfer Counter, and a 16-bit Control Word. The format of the DMA Control Blocks is shown in Table 13. The Transfer Count Register (TC) specifies the number of DMA transfers to be performed. Up to 64K byte transfers can be performed with automatic termination. The Control Word defines the channel's operation (see Figure 18). All registers may be modified or altered during any DMA activity. Any changes made to these registers will be reflected immediately in DMA operation.

Table 13. DMA Control Block Format

Register Name	Register Address	
	Ch. 0	Ch. 1
Control Word	CAH	DAH
Transfer Count	C8H	D8H
Destination Pointer (upper 4 bits)	C6H	D6H
Destination Pointer	C4H	D4H
Source Pointer (upper 4 bits)	C2H	D2H
Source Pointer	C0H	D0H

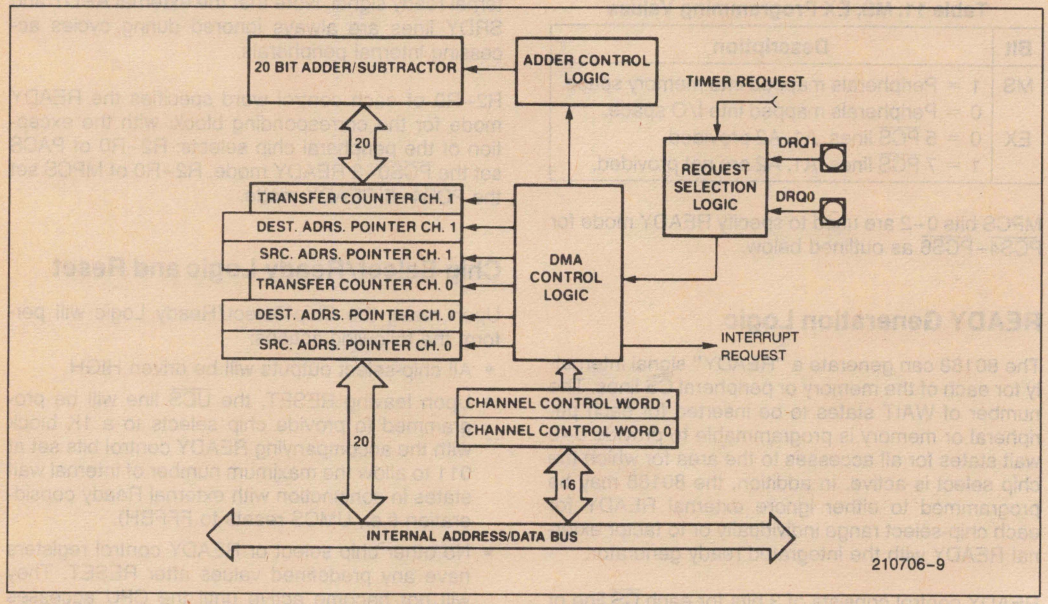


Figure 17. DMA Unit Block Diagram

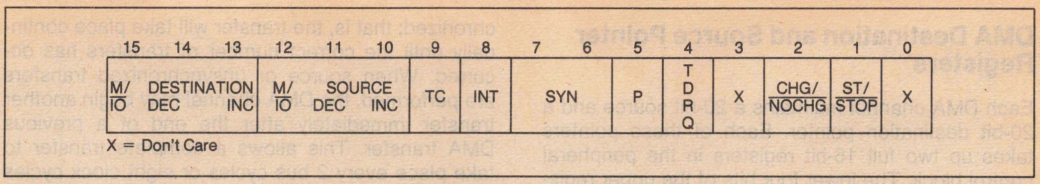


Figure 18. DMA Control Register

DMA Channel Control Word Register

Each DMA Channel Control Word determines the mode of operation for the particular 80188 DMA channel. This register specifies:

- the mode of synchronization;
- whether interrupts will be generated after the last transfer;
- whether DMA activity will cease after a programmed number of DMA cycles;
- the relative priority of the DMA channel with respect to the other DMA channel;
- whether the source pointer will be incremented, decremented, or maintained constant after each transfer;
- whether the source pointer addresses memory or I/O space;
- whether the destination pointer will be incremented, decremented, or maintained constant after each transfer; and
- whether the destination pointer will address memory or I/O space.

The DMA channel control registers may be changed while the channel is operating. However, any changes made during operation will affect the current DMA transfer.

DMA Control Word Bit Descriptions

ST/$\overline{\text{STOP}}$: Start/stop (1/0) Channel.

CHG/$\overline{\text{NOCHG}}$: Change/Do not change (1/0) ST/$\overline{\text{STOP}}$ bit. If this bit is set when writing to the control word, the ST/$\overline{\text{STOP}}$ bit will be programmed by the write to the control word. If this bit is cleared when writing the control word, the ST/$\overline{\text{STOP}}$ bit will not be altered. This bit is not stored; it will always be a 0 on read.

INT: Enable Interrupts to CPU on byte count termination.

TC: If set, DMA will terminate when the contents of the Transfer Count register reaches zero. The ST/$\overline{\text{STOP}}$ bit will also be reset at this point if TC is set. If this bit is cleared, the DMA unit will decrement the transfer count register for each DMA cycle, but the DMA transfer will not stop when the contents of the TC register reaches zero.

NOTE:

The ST bit will be cleared automatically when the contents of the TC register reaches zero regardless of the state of the TC bit.

SYN: (2 bits)
- 00 No Synchronization
- 01 Source Synchronization
- 10 Destination Synchronization
- 11 Unused

SOURCE:INC Increment source pointer by 1 after each transfer.

M/$\overline{\text{IO}}$ Source pointer is in M/$\overline{\text{IO}}$ space (1/0).

DEC Decrement source pointer by 1 after each transfer.

DEST: INC Increment destination pointer by 1 after each transfer.

M/$\overline{\text{IO}}$ Destination pointer is in M/$\overline{\text{IO}}$ space (1/0).

DEC Decrement destination pointer by 1 after each transfer.

P: Channel priority—relative to other channel.
- 0 low priority.
- 1 high priority.

Channels will alternate cycles if both set at same priority level.

TDRQ
- 0: Disable DMA requests from timer 2.
- 1: Enable DMA requests from timer 2.

Bit 3 Bit 3 is not used.

If both INC and DEC are specified for the same pointer, the pointer will remain constant after each cycle.

DMA Destination and Source Pointer Registers

Each DMA channel maintains a 20-bit source and a 20-bit destination pointer. Each of these pointers takes up two full 16-bit registers in the peripheral control block. The lower four bits of the upper register contain the upper four bits of the 20-bit physical address (see Figure 18a). These pointers may be individually incremented or decremented after each transfer. Each pointer may point into either memory or I/O space. Since the DMA channels can perform transfers to or from odd addresses, there is no restriction on values for the pointer registers.

DMA Transfer Count Register

Each DMA channel maintains a 16-bit transfer count register (TC). This register is decremented after every DMA cycle, regardless of the state of the TC bit in the DMA Control Register. If the TC bit in the DMA control word is set or unsynchronized transfers are programmed, DMA activity will terminate when the transfer count register reaches zero.

DMA Requests

Data transfers may be either source or destination synchronized, that is either the source of the data or the destination of the data may request the data transfer. In addition, DMA transfers may be unsynchronized; that is, the transfer will take place continually until the correct number of transfers has occurred. When source or unsynchronized transfers are performed, the DMA channel may begin another transfer immediately after the end of a previous DMA transfer. This allows a complete transfer to take place every 2 bus cycles or eight clock cycles (assuming no wait states). No prefetching occurs when destination synchronization is performed, however. Data will not be fetched from the source address until the destination device signals that it is ready to receive it. When destination synchronized transfers are requested, the DMA controller will relinquish control of the bus after every transfer. If no other bus activity is initiated, another DMA cycle will begin after two processor clocks. This is done to allow the destination device time to remove its request if another transfer is not desired. Since the DMA controller will relinquish the bus, the CPU can initiate a bus cycle. As a result, a complete bus cycle will often be inserted between destination synchronized transfers. These lead to the maximum DMA transfer rates shown in Table 14.

Table 14. Maximum DMA Transfer Rates

Type of Synchronization Selected	CPU Running	CPU Halted
Unsynchronized	1 MBytes/sec	1 MBytes/sec
Source Synch	1 MBytes/sec	1 MBytes/sec
Destination Synch	0.65 MBytes/sec	0.75 MBytes/sec

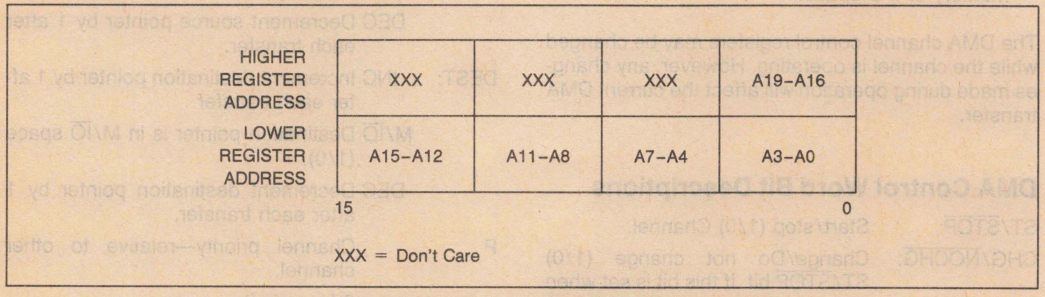


Figure 18a. DMA Memory Pointer Register Format

DMA Acknowledge

No explicit DMA acknowledge pulse is provided. Since both source and destination pointers are maintained, a read from a requesting source, or a write to a requesting destination, should be used as the DMA acknowledge signal. Since the chip-select lines can be programmed to be active for a given block of memory or I/O space, and the DMA pointers can be programmed to point to the same given block, a chip-select line could be used to indicate a DMA acknowledge.

DMA Priority

The DMA channels may be programmed such that one channel is always given priority over the other, or they may be programmed such as to alternate cycles when both have DMA requests pending. DMA cycles always have priority over internal CPU cycles except between locked memory accesses or word accesses the odd memory locations; however, an external bus hold takes priority over an internal DMA cycle. Because an interrupt request cannot suspend a DMA operation and the CPU cannot access memory during a DMA cycle, interrupt latency time will suffer during sequences of continuous DMA cycles. An NMI request, however, will cause all internal DMA activity to halt. This allows the CPU to quickly respond to the NMI request.

DMA Programming

DMA cycles will occur whenever the ST/STOP bit of the Control Register is set. If synchronized transfers are programmed, a DRQ must also have been generated. Therefore, the source and destination transfer pointers, and the transfer count register (if used) must be programmed before this bit is set.

Each DMA register may be modified while the channel is operating. If the CHG/NOCHG bit is cleared when the control register is written, the ST/STOP bit of the control register will not be modified by the write. If multiple channel registers are modified, it is recommended that a LOCKED string transfer be used to prevent a DMA transfer from occurring between updates to the channel registers.

DMA Channels and Reset

Upon RESET, the DMA channels will perform the following actions:

- The Start/Stop bit for each channel will be reset to STOP.
- Any transfer in progress is aborted.

TIMERS

The 80188 provides three internal 16-bit programmable timers (see Figure 19). Two of these are highly flexible and are connected to four external pins (2 per timer). They can be used to count external events, time external events, generate nonrepetitive waveforms, etc. The third timer is not connected to any external pins, and is useful for real-time coding and time delay applications. In addition, this third timer can be used as a prescaler to the other two, or as a DMA request source.

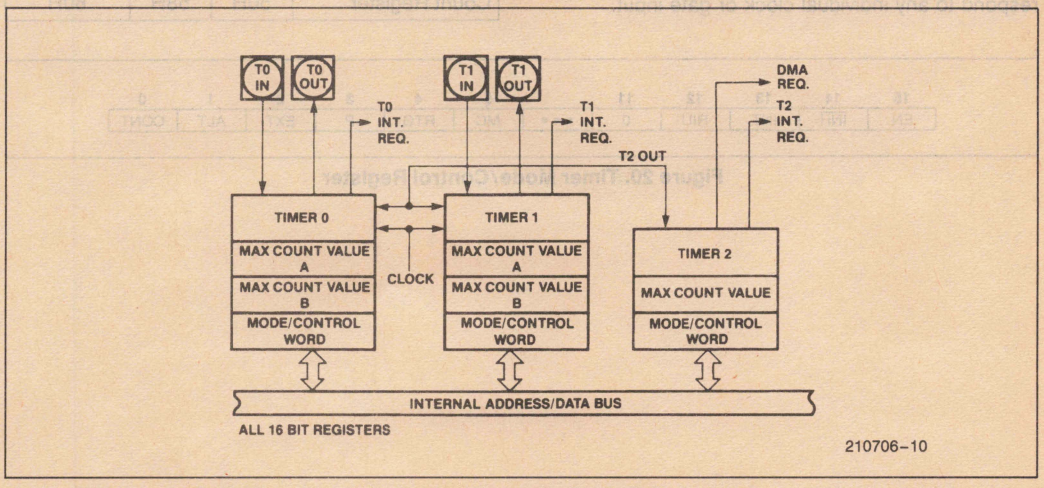


Figure 19. Timer Block Diagram

Timer Operation

The timers are controlled by 11 16-bit registers in the internal peripheral control block. The configuration of these registers is shown in Table 15. The count register contains the current value of the timer. It can be read or written at any time independent of whether the timer is running or not. The value of this register will be incremented for each timer event. Each of the timers is equipped with a MAX COUNT register, which defines the maximum count the timer will reach. After reaching the MAX COUNT register value, the timer count value will reset to zero during that same clock, i.e., the maximum count value is never stored in the count register itself. Timers 0 and 1 are, in addition, equipped with a second MAX COUNT register, which enables the timers to alternate their count between two different MAX COUNT values programmed by the user. If a single MAX COUNT register is used, the timer output pin will switch LOW for a single clock, 2 clocks after the maximum count value has been reached. In the dual MAX COUNT register mode, the output pin will indicate which MAX COUNT register is currently in use, thus allowing nearly complete freedom in selecting waveform duty cycles. For the timers with two MAX COUNT registers, the RIU bit in the control register determines which is used for the comparison.

Each timer gets serviced every fourth CPU-clock cycle, and thus can operate at speeds up to one-quarter the internal clock frequency (one-eighth the crystal rate). External clocking of the timers may be done at up to a rate of one-quarter of the internal CPU-clock rate (2 MHz for an 8 MHz CPU clock). Due to internal synchronization and pipelining of the timer circuitry, a timer output may take up to 6 clocks to respond to any individual clock or gate input.

Since the count registers and the maximum count registers are all 16 bits wide, 16 bits of resolution are provided. Any Read or Write access to the timers will add one wait state to the minimum four-clock bus cycle, however. This is needed to synchronize and coordinate the internal data flows between the internal timers and the internal bus.

The timers have several programmable options.

- All three timers can be set to halt or continue on a terminal count.
- Timers 0 and 1 can select between internal and external clocks, alternate between MAX COUNT registers and be set to retrigger on external events.
- The timers may be programmed to cause an interrupt on terminal count.

These options are selectable via the timer mode/control word.

Timer Mode/Control Register

The mode/control register (see Figure 20) allows the user to program the specific mode of operation or check the current programmed status for any of the three integrated timers.

Table 15. Timer Control Block Format

Register Name	Register Offset		
	Tmr. 0	Tmr. 1	Tmr. 2
Mode/Control Word	56H	5EH	66H
Max Count B	54H	5CH	not present
Max Count A	52H	5AH	62H
Count Register	50H	58H	60H

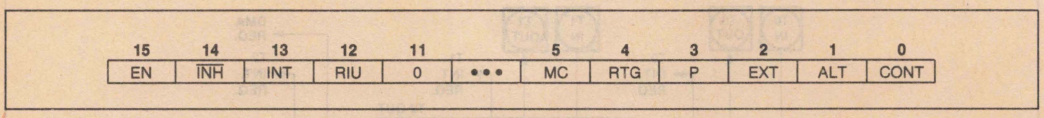


Figure 20. Timer Mode/Control Register

ALT

The ALT bit determines which of two MAX COUNT registers is used for count comparison. If ALT = 0, register A for that timer is always used, while if ALT = 1, the comparison will alternate between register A and register B when each maximum count is reached. This alternation allows the user to change one MAX COUNT register while the other is being used, and thus provides a method of generating non-repetitive waveforms. Square waves and pulse outputs of any duty cycle are a subset of available signals obtained by not changing the final count registers. The ALT bit also determines the function of the timer output pin. If ALT is zero, the output pin will go LOW for one clock, the clock after the maximum count is reached. If ALT is one, the output pin will reflect the current MAX COUNT register being used (0/1 for B/A).

CONT

Setting the CONT bit causes the associated timer to run continuously, while resetting it causes the timer to halt upon maximum count. If CONT = 0 and ALT = 1, the timer will count to the MAX COUNT register A value, reset, count to the register B value, reset, and halt.

EXT

The external bit selects between internal and external clocking for the timer. The external signal may be asynchronous with respect to the 80188 clock. If this bit is set, the timer will count LOW-to-HIGH transitions on the input pin. If cleared, it will count an internal clock while using the input pin for control. In this mode, the function of the external pin is defined by the RTG bit. The maximum input to output transition latency time may be as much as 6 clocks. However, clock inputs may be pipelined as closely together as every 4 clocks without losing clock pulses.

P

The prescaler bit is ignored unless internal clocking has been selected (EXT = 0). If the P bit is a zero, the timer will count at one-fourth the internal CPU clock rate. If the P bit is a one, the output of timer 2 will be used as a clock for the timer. Note that the user must initialize and start timer 2 to obtain the prescaled clock.

RTG

Retrigger bit is only active for internal clocking (EXT = 0). In this case it determines the control function provided by the input pin.

If RTG = 0, the input level gates the internal clock on and off. If the input pin is HIGH, the timer will count; if the input pin is LOW, the timer will hold its value. As indicated previously, the input signal may be asynchronous with respect to the 80188 clock.

When RTG = 1, the input pin detects LOW-to-HIGH transitions. The first such transition starts the timer running, clearing the timer value to zero on the first clock, and then incrementing thereafter. Further transitions on the input pin will again reset the timer to zero, from which it will start counting up again. If CONT = 0, when the timer has reached maximum count, the EN bit will be cleared, inhibiting further timer activity.

EN

The enable bit provides programmer control over the timer's RUN/HALT status. When set, the timer is enabled to increment subject to the input pin constraints in the internal clock mode (discussed previously). When cleared, the timer will be inhibited from counting. All input pin transitions during the time EN is zero will be ignored. If CONT is zero, the EN bit is automatically cleared upon maximum count.

INH

The inhibit bit allows for selective updating of the enable (EN) bit. If INH is a one during the write to the mode/control word, then the state of the EN bit will be modified by the write. If INH is a zero during the write, the EN bit will be unaffected by the operation. This bit is not stored; it will always be a 0 on a read.

INT

When set, the INT bit enables interrupts from the timer, which will be generated on every terminal count. If the timer is configured in dual MAX COUNT register mode, an interrupt will be generated each time the value in MAX COUNT register A is reached, and each time the value in MAX COUNT register B is reached. If this enable bit is cleared after the interrupt request has been generated, but before a pending interrupt is serviced, the interrupt request will still be in force. (The request is latched in the Interrupt Controller.)

MC

The Maximum Count bit is set whenever the timer reaches its final maximum count value. If the timer is configured in dual MAX COUNT register mode, this bit will be set each time the value in MAX COUNT register A is reached, and each time the value in MAX COUNT register B is reached. This bit is set

regardless of the timer's interrupt-enable bit. The MC bit gives the user the ability to monitor timer status through software instead of through interrupts. Programmer intervention is required to clear this bit.

RIU

The Register In Use bit indicates which MAX COUNT register is currently being used for comparison to the timer count value. A zero value indicates register A. The RIU bit cannot be written, i.e., its value is not affected when the control register is written. It is always cleared when the ALT bit is zero.

Not all mode bits are provided for timer 2. Certain bits are hardwired as indicated below:

ALT = 0, EXT = 0, P = 0, RTG = 0, RIU = 0

Count Registers

Each of the three timers has a 16-bit count register. The current contents of this register may be read or written by the processor at any time. If the register is written into while the timer is counting, the new value will take effect in the current count cycle.

Max Count Registers

Timers 0 and 1 have two MAX COUNT registers, while timer 2 has a single MAX COUNT register. These contain the number of events the timer will count. In timers 0 and 1, the MAX COUNT register used can alternate between the two max count values whenever the current maximum count is reached. The condition which causes a timer to reset is equivalent between the current count value and the max count being used. This means that if the count is changed to be above the max count value, or if the max count value is changed to be below the current value, the timer will not reset to zero, but rather will count to its maximum value, "wrap around" to zero, then count until the max count is reached.

Timers and Reset

Upon RESET, the Timers will perform the following actions:

- All EN (Enable) bits are reset preventing timer counting.
- All SEL (Select) bits are reset to zero. This selects MAX COUNT register A, resulting in the Timer Out pins going HIGH upon RESET.

INTERRUPT CONTROLLER

The 80188 can receive interrupts from a number of sources, both internal and external. The internal interrupt controller serves to merge these requests on a priority basis, for individual service by the CPU.

Internal interrupt sources (Timers and DMA channels) can be disabled by their own control registers or by mask bits within the interrupt controller. The 80188 interrupt controller has its own control register that set the mode of operation for the controller.

The interrupt controller will resolve priority among requests that are pending simultaneously. Nesting is provided so interrupt service routines for lower priority interrupts may themselves be interrupted by higher priority interrupts. A block diagram of the interrupt controller is shown in Figure 21.

The interrupt controller has a special iRMX 86 compatibility mode that allows the use of the 80188 within the iRMX 86 operating system interrupt structure. The controller is set in this mode by setting bit 14 in the peripheral control block relocation register (see iRMX 86 Compatibility Mode section). In this mode, the internal 80188 interrupt controller functions as a "slave" controller to an external "master" controller. Special initialization software must be included to properly set up the 80188 interrupt controller in iRMX 86 mode.

NON-iRMX™ MODE OPERATION

Interrupt Controller External Interface

For external interrupt sources, five dedicated pins are provided. One of these pins is dedicated to NMI, non-maskable interrupt. This is typically used for power-fail interrupts, etc. The other four pins may function either as four interrupt input lines with internally generated interrupt vectors, as an interrupt line and an interrupt acknowledge line (called the "cascade mode") along with two other input lines with internally generated interrupt vectors, or as two interrupt input lines and two dedicated interrupt acknowledge output lines. When the interrupt lines are configured in cascade mode, the 80188 interrupt controller will not generate internal interrupt vectors.

External sources in the cascade mode use externally generated interrupt vectors. When an interrupt is acknowledged, two $\overline{\text{INTA}}$ cycles are initiated and the vector is read into the 80188 on the second cycle. The capability to interface to external 8259A programmable interrupt controllers is thus provided when the inputs are configured in cascade mode.

Interrupt Controller Modes of Operation

The basic modes of operation of the interrupt controller in non-iRMX™ mode are similar to the 8259A. The interrupt controller responds indentically to internal interrupts in all three modes: the difference is only in the interpretation of function of the four external interrupt pins. The interrupt controller is set into one of these three modes by programming the correct bits in the INT0 and INT1 control registers. The modes of interrupt controller operation are as follows:

FULLY NESTED MODE

When in the fully nested mode four pins are used as direct interrupt requests. The vectors for these four inputs are generated internally. An in-service bit is provided for every interrupt source. If a lower-priority device requests an interrupt while the in-service bit (IS) is set, no interrupt will be generated by the interrupt controller. In addition, if another interrupt request occurs from the same interrupt source while the in-service bit is set, no interrupt will be generated by the interrupt controller. This allows interrupt service routines to operate with interrupts enabled without being themselves interrupted by lower-priority interrupts. Since interrupts are enabled, higher-priority interrupts will be serviced.

When a service routine is completed, the proper IS bit must be reset by writing the proper pattern to the EOI register. This is required to allow subsequent interrupts from this interrupt source and to allow servicing of lower-priority interrupts. An EOI command is issued at the end of the service routine just before the issuance of the return from interrupt instruction. If the fully nested structure has been upheld, the next highest-priority source with its IS bit set is then serviced.

CASCADE MODE

The 80188 has four interrupt pins and two of them have dual functions. In the fully nested mode the four pins are used as direct interrupt inputs and the corresponding vectors are generated internally. In the cascade mode, the four pins are configured into interrupt input-dedicated acknowledge signal pairs. The interconnection is shown in Figure 22. INT0 is an interrupt input interfaced to an 8259A, while INT2/$\overline{\text{INTA0}}$ serves as the dedicated interrupt acknowledge signal to that peripheral. The same is true for INT1 and INT3/$\overline{\text{INTA1}}$. Each pair can selectively be placed in the cascade or non-cascade mode by programming the proper value into INT0 and INT1 control registers. The use of the dedicated acknowledge signals eliminates the need for the use of external logic to generate $\overline{\text{INTA}}$ and device select signals.

The primary cascade mode allows the capability to serve up to 128 external interrupt sources through the use of external master and slave 8259As. Three levels of priority are created, requiring priority resolution in the 80188 interrupt controller, the master 8259As, and the slave 8259As. If an external interrupt is serviced, one IS bit is set at each of these levels. When the interrupt service routine is completed, up to three end-of-interrupt commands must be issued by the programmer.

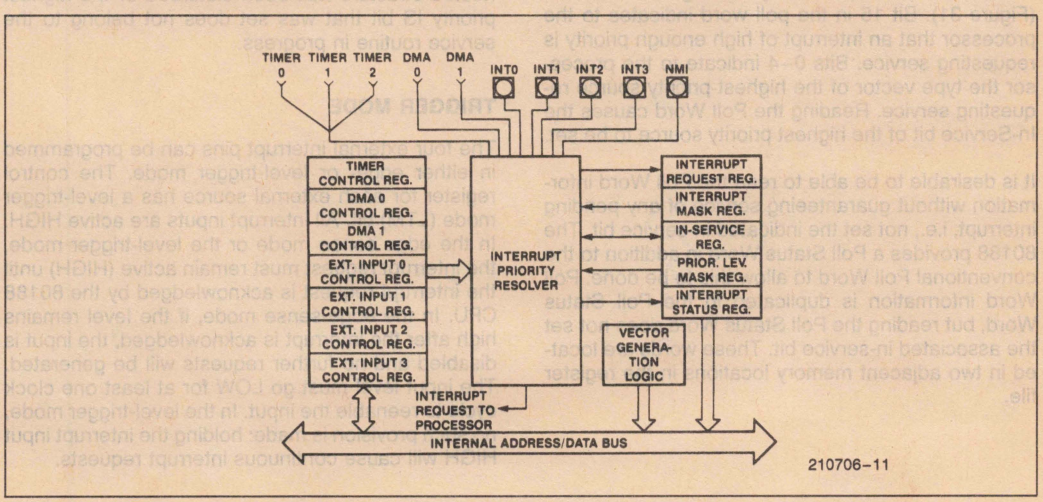


Figure 21. Interrupt Controller Block Diagram

SPECIAL FULLY NESTED MODE

This mode is entered by setting the SFNM bit in INT0 or INT1 control register. It enables complete nestability with external 8259A masters. Normally, an interrupt request from an interrupt source will not be recognized unless the in-service bit for that source is reset. If more than one interrupt source is connected to an external interrupt controller, all of the interrupts will be funneled through the same 80188 interrupt request pin. As a result, if the external interrupt controller receives a higher-priority interrupt, its interrupt will not be recognized by the 80188 controller until the 80188 in-service bit is reset. In special fully nested mode, the 80188 interrupt controller will allow interrupts from an external pin regardless of the state of the in-service bit for an interrupt source in order to allow multiple interrupts from a single pin. An in-service bit will continue to be set, however, to inhibit interrupts from other lower-priority 80188 interrupt sources.

Special procedures should be followed when resetting IS bits at the end of interrupt service routines. Software polling of the external master's IS register is required to determine if there is more than one bit set. If so, the IS bit in the 80188 remains active and the next interrupt service routine is entered.

Operation in a Polled Environment

The controller may be used in a polled mode if interrupts are undesirable. When polling, the processor disables interrupts and then polls the interrupt controller whenever it is convenient. Polling the interrupt controller is accomplished by reading the Poll Word (Figure 31). Bit 15 in the poll word indicates to the processor that an interrupt of high enough priority is requesting service. Bits 0–4 indicate to the processor the type vector of the highest-priority source requesting service. Reading the Poll Word causes the In-Service bit of the highest priority source to be set.

It is desirable to be able to read the Poll Word information without guaranteeing service of any pending interrupt, i.e., not set the indicated in-service bit. The 80188 provides a Poll Status Word in addition to the conventional Poll Word to allow this to be done. Poll Word information is duplicated in the Poll Status Word, but reading the Poll Status Word does not set the associated in-service bit. These words are located in two adjacent memory locations in the register file.

Non-iRMX™ Mode Features

PROGRAMMABLE PRIORITY

The user can program the interrupt sources into any of eight different priority levels. The programming is done by placing a 3-bit priority level (0–7) in the control register of each interrupt source. (A source with a priority level of 4 has higher priority over all priority levels from 5 to 7. Priority registers containing values lower than 4 have greater priority). All interrupt sources have preprogrammed default priority levels (see Table 4).

If two requests with the same programmed priority level are pending at once, the priority ordering scheme shown in Table 4 is used. If the serviced interrupt routine reenables interrupts, it allows other requests to be serviced.

END-OF-INTERRUPT COMMAND

The end-of-interrupt (EOI) command is used by the programmer to reset the In-Service (IS) bit when an interrupt service routine is completed. The EOI command is issued by writing the proper pattern to the EOI register. There are two types of EOI commands, specific and nonspecific. The nonspecific command does not specify which IS bit is reset. When issued, the interrupt controller automatically resets the IS bit of the highest priority source with an active service routine. A specific EOI command requires that the programmer send the interrupt vector type to the interrupt controller indicating which source's IS bit is to be reset. This command is used when the fully nested structure has been disturbed or the highest priority IS bit that was set does not belong to the service routine in progress.

TRIGGER MODE

The four external interrupt pins can be programmed in either edge- or level-trigger mode. The control register for each external source has a level-trigger mode (LTM) bit. All interrupt inputs are active HIGH. In the edge sense mode or the level-trigger mode, the interrupt request must remain active (HIGH) until the interrupt request is acknowledged by the 80188 CPU. In the edge-sense mode, if the level remains high after the interrupt is acknowledged, the input is disabled and no further requests will be generated. The input level must go LOW for at least one clock cycle to reenable the input. In the level-trigger mode, no such provision is made: holding the interrupt input HIGH will cause continuous interrupt requests.

INTERRUPT VECTORING

The 80188 Interrupt Controller will generate interrupt vectors for the integrated DMA channels and the integrated Timers. In addition, the Interrupt Controller will generate interrupt vectors for the external interrupt lines if they are not configured in Cascade or Special Fully Nested Mode. The interrupt vectors generated are fixed and cannot be changed (see Table 4).

Interrupt Controller Registers

The Interrupt Controller register model is shown in Figure 23. It contains 15 registers. All registers can both be read or written unless specified otherwise.

IN-SERVICE REGISTER

This register can be read from or written into. The format is shown in Figure 24. It contains the In-Service bit for each of the interrupt sources. The In-Service bit is set to indicate that a source's service routine is in progress. When an In-Service bit is set, the interrupt controller will not generate interrupts to the CPU when it receives interrupt requests from devices with a lower programmed priority level. The TMR bit is the In-Service bit for all three timers; the D0 and D1 bits are the In-Service bits for the two DMA channels; the I0–I3 are the In-Service bits for the external interrupt pins. The IS bit is set when the processor acknowledges an interrupt request either by an interrupt acknowledge or by reading the poll register. The IS bit is reset at the end of the interrupt service routine by an end-of-interrupt command issued by the CPU.

INTERRUPT REQUEST REGISTER

The internal interrupt sources have interrupt request bits inside the interrupt controller. The format of this register is shown in Figure 24. A read from this register yields the status of these bits. The TMR bit is the logical OR of all timer interrupt requests. D0 and D1 are the interrupt request bits for the DMA channels.

The state of the external interrupt input pins is also indicated. The state of the external interrupt pins is not a stored condition inside the interrupt controller, therefore the external interrupt bits cannot be written. The external interrupt request bits show exactly when an interrupt request is given to the interrupt controller, so if edge-triggered mode is selected, the bit in the register will be HIGH only after an inactive-to-active transition. For internal interrupt sources, the register bits are set when a request arrives and are reset when the processor acknowledges the requests.

MASK REGISTER

This is a 16-bit register that contains a mask bit for each interrupt source. The format for this register is shown in Figure 24. A one in a bit position corresponding to a particular source serves to mask the source from generating interrupts. These mask bits are the exact same bits which are used in the individual control registers; programming a mask bit using the mask register will also change this bit in the individual control registers, and vice versa.

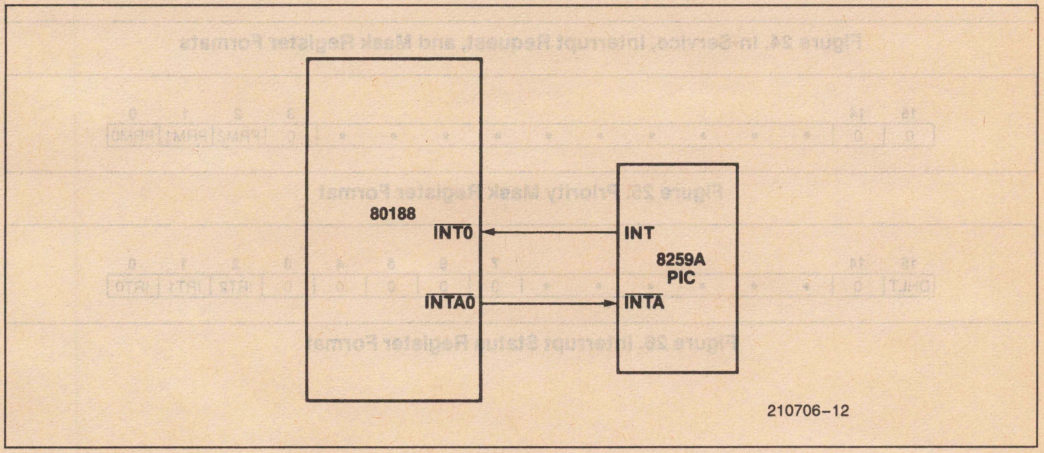


Figure 22. Cascade Mode Interrupt Connection

	OFFSET
INT3 CONTROL REGISTER	3EH
INT2 CONTROL REGISTER	3CH
INT1 CONTROL REGISTER	3AH
INT0 CONTROL REGISTER	38H
DMA 1 CONTROL REGISTER	36H
DMA 0 CONTROL REGISTER	34H
TIMER CONTROL REGISTER	32H
INTERRUPT STATUS REGISTER	30H
INTERRUPT REQUEST REGISTER	2EH
IN-SERVICE REGISTER	2CH
PRIORITY MASK REGISTER	2AH
MASK REGISTER	28H
POLL STATUS REGISTER	26H
POLL REGISTER	24H
EOI REGISTER	22H

Figure 23. Interrupt Controller Registers (Non-iRMX™ 86 Mode)

PRIORITY MASK REGISTER

This register is used to mask all interrupts below particular interrupt priority levels. The format of this register is shown in Figure 25. The code in the lower three bits of this register inhibits interrupts of priority lower (a higher priority number) than the code specified. For example, 100 written into this register masks interrupts of level five (101), six (110), and seven (111). The register is reset to seven (111) upon RESET so all interrupts are unmasked.

INTERRUPT STATUS REGISTER

This register contains general interrupt controller status information. The format of this register is shown in Figure 26. The bits in the status register have the following functions:

DHLT: DMA Halt Transfer; setting this bit halts all DMA transfers. It is automatically set whenever a non-maskable interrupt occurs, and it is reset when an IRET instruction is executed. The purpose of this bit is to allow prompt service of all non-maskable interrupts. This bit may also be set by the CPU.

IRTx: These three bits represent the individual timer interrupt request bits. These bits are used to differentiate the timer interrupts, since the timer IR bit in the interrupt request register is the "OR" function of all timer interrupt requests. Note that setting any one of these three bits initiates an interrupt request to the interrupt controller.

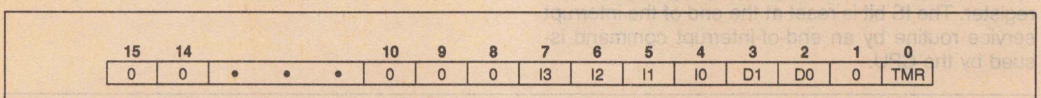


Figure 24. In-Service, Interrupt Request, and Mask Register Formats

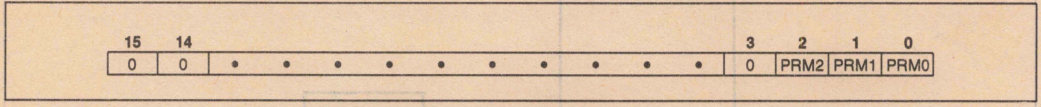


Figure 25. Priority Mask Register Format

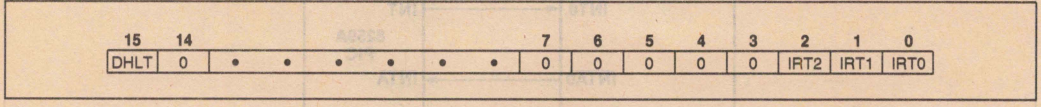


Figure 26. Interrupt Status Register Format

TIMER, DMA 0, 1; CONTROL REGISTERS

These registers are the control words for all the internal interrupt sources. The format for these registers is shown in Figure 27. The three bit positions PR0, PR1, and PR2 represent the programmable priority level of the interrupt source. The MSK bit inhibits its interrupt requests from the interrupt source. The MSK bits in the individual control registers are the exact same bits as are in the Mask Register; modifying them in the individual control registers will also modify them in the Mask Register, and vice versa.

INT0-INT3 CONTROL REGISTERS

These registers are the control words for the four external input pins. Figure 28 shows the format of the INT0 and INT1 Control registers; Figure 29 shows the format of the INT2 and INT3 Control registers. In cascade mode or special fully nested mode, the control words for INT2 and INT3 are not used.

The bits in the various control registers are encoded as follows:

PRO-2: Priority programming information. Highest priority = 000, lowest priority = 111.

LTM: Level-trigger mode bit. 1 = level-triggered; 0 = edge-triggered. Interrupt Input levels are active high. In level-triggered mode, an interrupt is generated whenever the external line is high. In edge-triggered mode, an interrupt will be generated only when this level is preceded by an inactive-to-active transition on the line. In both cases, the level must remain active until the interrupt is acknowledged.

MSK: Mask bit, 1 = mask; 0 = non-mask.

C: Cascade mode bit, 1 = cascade; 0 = direct.

SFNM: Special fully nested mode bit, 1 = SFNM.

EOI REGISTER

The end of the interrupt register is a command register which can only be written into. The format of this register is shown in Figure 30. It initiates an EOI command when written to by the 80188 CPU.

The bits in the EOI register are encoded as follows:

S_x: Encoded information that specifies an interrupt source vector type as shown in Table 4. For example, to reset the In-Service bit for DMA channel 0, these bits should be set to 01010, since the vector type for DMA channel 0 is 10. Note that to reset the single In-Service bit for any of the three timers, the vector type for timer 0 (8) should be written in this register.

NSPEC/SPEC: A bit that determines the type of EOI command. Nonspecific = 1, Specific = 0.

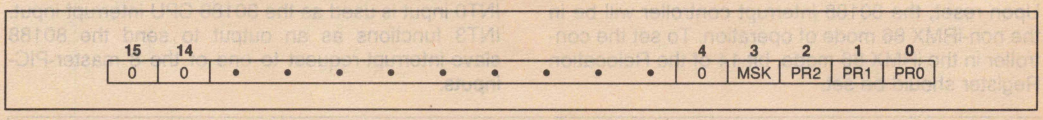


Figure 27. Timer/DMA Control Register Formats

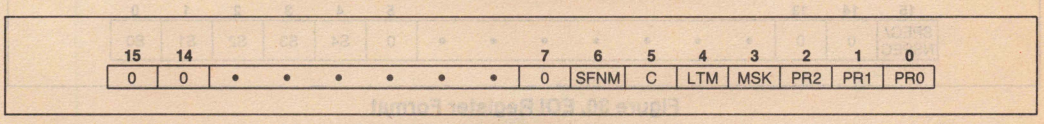


Figure 28. INT0/INT1 Control Register Formats

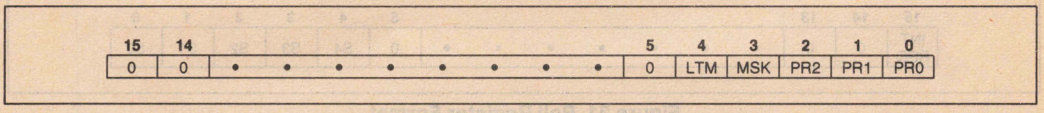


Figure 29. INT2/INT3 Control Register Formats

POLL AND POLL STATUS REGISTERS

These registers contain polling information. The format of these registers is shown in Figure 31. They can only be read. Reading the Poll register constitutes a software poll. This will set the IS bit of the highest priority pending interrupt. Reading the poll status register will not set the IS bit of the highest priority pending interrupt; only the status of pending interrupts will be provided.

Encoding of the Poll and Poll Status register bits are as follows:

S_x: Encoded information that indicates the vector type of the highest priority interrupting source. Valid only when INTREQ = 1.

INTREQ: This bit determines if an interrupt request is present. Interrupt Request = 1; no Interrupt Request = 0.

iRMX™ 86 COMPATIBILITY MODE

This mode allows iRMX 86-80188 compatibility. The interrupt model of iRMX 86 requires one master and multiple slave 8259As in cascaded fashion. When iRMX mode is used, the internal 80188 interrupt controller will be used as a slave controller to an external master interrupt controller. The internal 80188 resources will be monitored through the internal interrupt controller, while the external controller functions as the system master interrupt controller.

Upon reset, the 80188 interrupt controller will be in the non-iRMX 86 mode of operation. To set the controller in the iRMX 86 mode, bit 14 of the Relocation Register should be set.

Because of pin limitations caused by the need to interface to an external 8259A master, the internal interrupt controller will no longer accept external inputs. There are however, enough 80188 interrupt controller inputs (internally) to dedicate one to each timer. In this mode, each timer interrupt source has its own mask bit, IS bit, and control word.

The iRMX 86 operating system requires peripherals to be assigned fixed priority levels. This is incompatible with the normal operation of the 80188 interrupt controller. Therefore, the initialization software must program the proper priority levels for each source. The required priority levels for the internal interrupt sources in iRMX mode are shown in Table 16.

Table 16. Internal Source Priority Level

Priority Level	Interrupt Source
0	Timer 0
1	(reserved)
2	DMA 0
3	DMA 1
4	Timer 1
5	Timer 2

These level assignments must remain fixed in the iRMX 86 mode of operation.

iRMX™ 86 Mode External Interface

The configuration of the 80188 with respect to an external 8259A master is shown in Figure 32. The INT0 input is used as the 80188 CPU interrupt input. INT3 functions as an output to send the 80188 slave-interrupt-request to one of the 8 master-PIC-inputs.

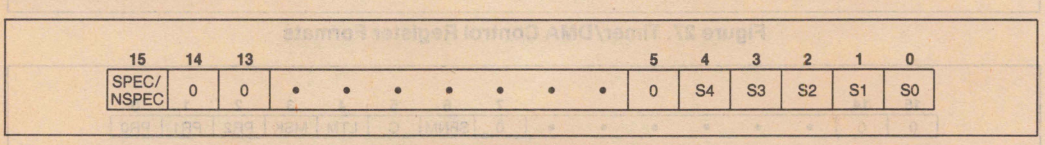


Figure 30. EOI Register Format

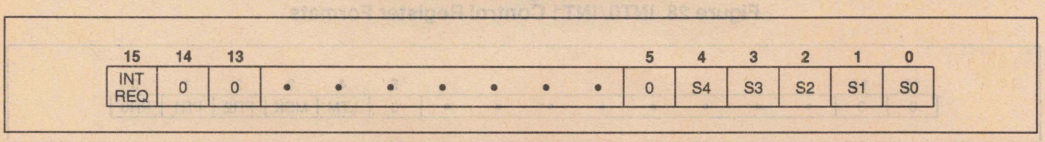


Figure 31. Poll Register Format

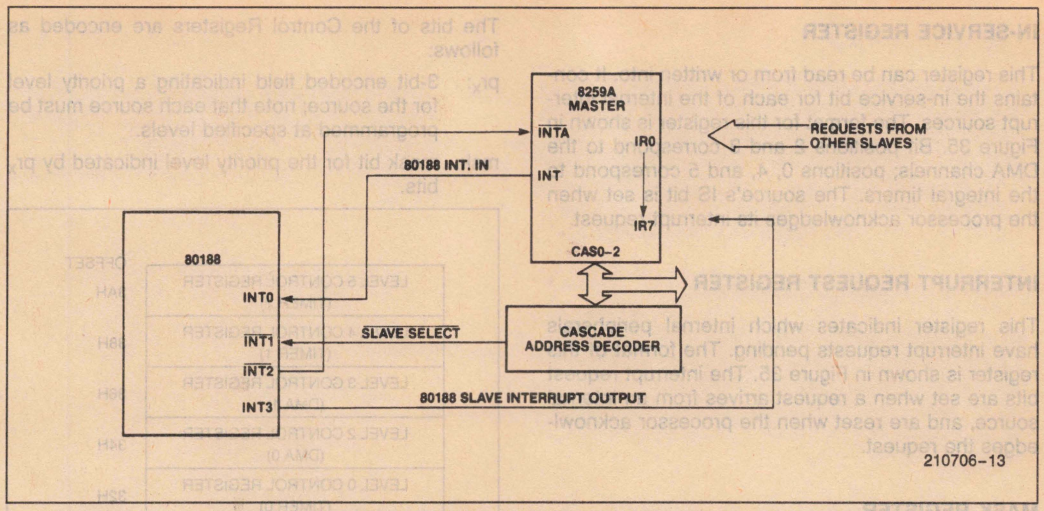


Figure 32. iRMX™ 86 Interrupt Controller Interconnection

Correct master-slave interface requires decoding of the slave addresses (CAS0–2). Slave 8259As do this internally. Because of pin limitations, the 80188 slave address will have to be decoded externally. $\overline{INT1}$ is used as a slave-select input. Note that the slave vector address is transferred internally, but the READY input must be supplied externally.

$\overline{INT2}$ is used as an acknowledge output, suitable to drive the $\overline{INTA}$ input of an 8259A.

Interrupt Nesting

iRMX 86 mode operation allows nesting of interrupt requests. When an interrupt is acknowledged, the priority logic masks off all priority levels except those with equal or higher priority.

Vector Generation in the iRMX™ 86 Mode

Vector generation in iRMX mode is exactly like that of an 8259A slave. The interrupt controller generates an 8-bit vector which the CPU multiplies by four and uses as an address into a vector table. The significant five bits of the vector are user-programmable while the lower three bits are generated by the priority logic. These bits represent the encoding of the priority level requesting service. The significant five bits of the vector are programmed by writing to the Interrupt Vector register at offset 20H.

Specific End-of-Interrupt

In iRMX mode the specific EOI command operates to reset an in-service bit of a specific priority. The user supplies a 3-bit priority-level value that points to an in-service bit to be reset. The command is executed by writing the correct value in the Specific EOI register at offset 22H.

Interrupt Controller Registers in the iRMX™ 86 Mode

All control and command registers are located inside the internal peripheral control block. Figure 33 shows the offsets of these registers.

END-OF-INTERRUPT REGISTER

The end-of-interrupt register is a command register which can only be written. The format of this register is shown in Figure 34. It initiates an EOI command when written by the 80188 CPU.

The bits in the EOI register are encoded as follows:

L_X: Encoded value indicating the priority of the IS bit to be reset.

IN-SERVICE REGISTER

This register can be read from or written into. It contains the in-service bit for each of the internal interrupt sources. The format for this register is shown in Figure 35. Bit positions 2 and 3 correspond to the DMA channels; positions 0, 4, and 5 correspond to the integral timers. The source's IS bit is set when the processor acknowledges its interrupt request.

INTERRUPT REQUEST REGISTER

This register indicates which internal peripherals have interrupt requests pending. The format of this register is shown in Figure 35. The interrupt request bits are set when a request arrives from an internal source, and are reset when the processor acknowledges the request.

MASK REGISTER

The register contains a mask bit for each interrupt source. The format for this register is shown in Figure 35. If the bit in this register corresponding to a particular interrupt source is set, any interrupts from that source will be masked. These mask bits are exactly the same bits which are used in the individual control registers, i.e., changing the state of a mask bit in this register will also change the state of the mask bit in the individual interrupt control register corresponding to the bit.

CONTROL REGISTERS

These registers are the control words for all the internal interrupt sources. The format of these registers is shown in Figure 36. Each of the timers and both of the DMA channels have their own Control Register.

The bits of the Control Registers are encoded as follows:

pr_x: 3-bit encoded field indicating a priority level for the source; note that each source must be programmed at specified levels.

msk: mask bit for the priority level indicated by pr_x bits.

	OFFSET
LEVEL 5 CONTROL REGISTER (TIMER 2)	3AH
LEVEL 4 CONTROL REGISTER (TIMER 1)	38H
LEVEL 3 CONTROL REGISTER (DMA 1)	36H
LEVEL 2 CONTROL REGISTER (DMA 0)	34H
LEVEL 0 CONTROL REGISTER (TIMER 0)	32H
INTERRUPT STATUS REGISTER	30H
INTERRUPT REQUEST REGISTER	2EH
IN-SERVICE REGISTER	2CH
PRIORITY-LEVEL MASK REGISTER	2AH
MASK REGISTER	28H
SPECIFIC EOI REGISTER	22H
INTERRUPT VECTOR REGISTER	20H

Figure 33. Interrupt Controller Registers (iRMX™ 86 Mode)

15	14	13	•	•	•	•	8	7	6	5	4	3	2	1	0
0	0	0					0	0	0	0	0	0	L2	L1	L0

Figure 34. Specific EOI Register Format

15	14	13	•	•	•	•	8	7	6	5	4	3	2	1	0
0	0	0					0	0	0	TMR2	TMR1	D1	D0	0	TMR0

Figure 35. In-Service, Interrupt Request, and Mask Register Format

INTERRUPT VECTOR REGISTER

This register provides the upper five bits of the interrupt vector address. The format of this register is shown in Figure 37. The interrupt controller itself provides the lower three bits of the interrupt vector as determined by the priority level of the interrupt request.

The format of the bits in this register is:

t_x: 5-bit field indicating the upper five bits of the vector address.

PRIORITY-LEVEL MASK REGISTER

This register indicates the lowest priority-level interrupt which will be serviced.

The encoding of the bits in this register is:

m_x: 3-bit encoded field indication priority-level value. All levels of lower priority will be masked.

INTERRUPT STATUS REGISTER

This register is defined exactly as in non-iRMX mode (See Figure 26).

Interrupt Controller and Reset

Upon RESET, the interrupt controller will perform the following actions:

- All SFNM bits reset to 0, implying Fully Nested Mode.
- All PR bits in the various control registers set to 1. This places all sources at lowest priority (level 111).
- All LTM bits reset to 0, resulting in edge-sense mode.
- All Interrupt Service bits reset to 0.
- All Interrupt Request bits reset to 0.
- All MSK (Interrupt Mask) bits set to 1 (mask).
- All C (Cascade) bits reset to 0 (non-cascade).
- All PRM (Priority Mask) bits set to 1, implying no levels masked.
- Initialized to non-iRMX 86 mode.

15	14	13	•	•	•	•	8	7	6	5	4	3	2	1	0
0	0	0	•	•	•	•	0	0	0	0	0	MSK	PR2	PR1	PR0

Figure 36. Control Word Format

15	14	13	•	•	•	•	8	7	6	5	4	3	2	1	0
0	0	0	•	•	•	•	0	t4	t3	t2	t1	t0	0	0	0

Figure 37. Interrupt Vector Register Format

15	14	13	•	•	•	•	8	7	6	5	4	3	2	1	0
0	0	0	•	•	•	•	0	0	0	0	0	0	m2	m1	m0

Figure 38. Priority Level Mask Register

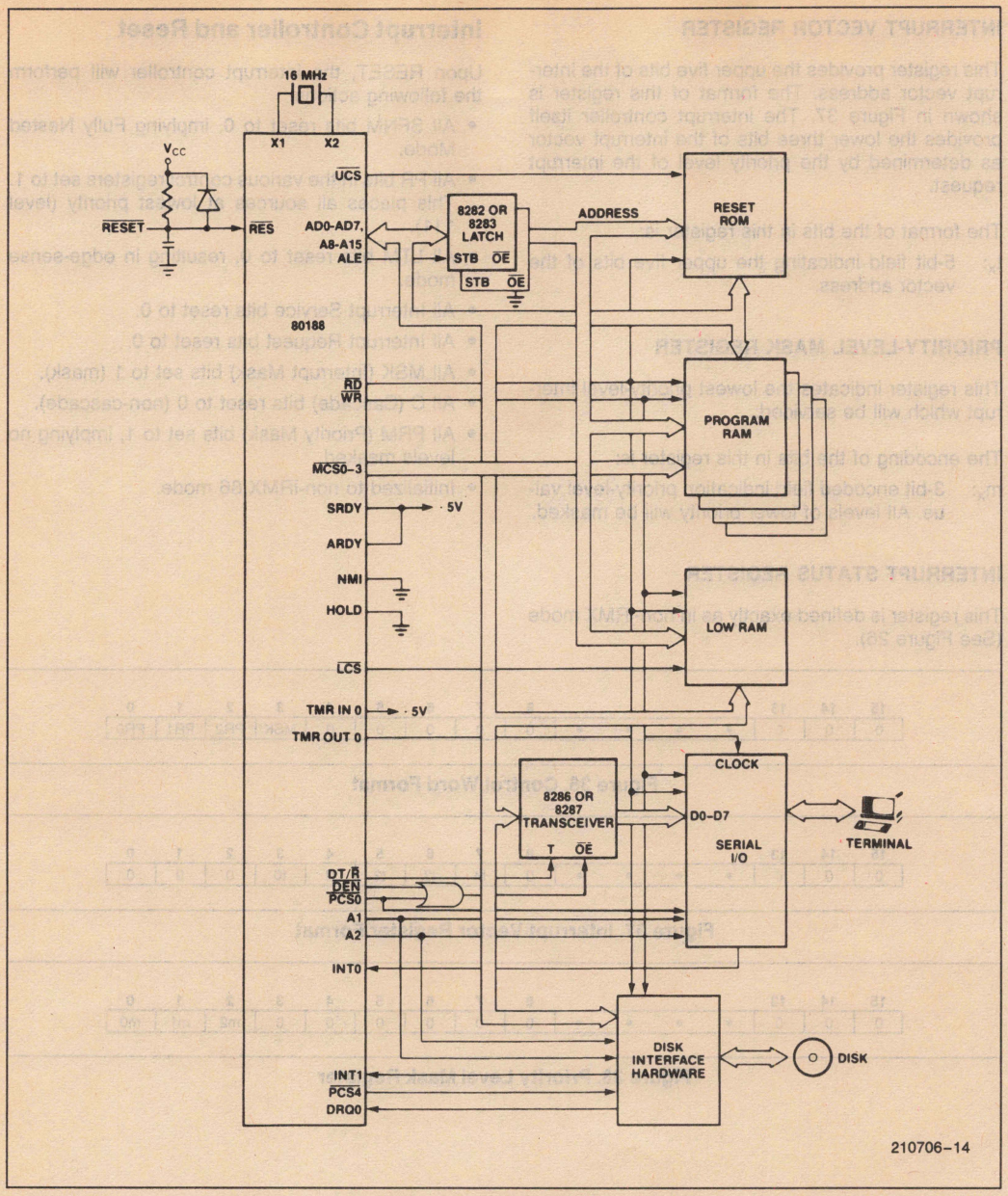


Figure 39. Typical 80188 Computer

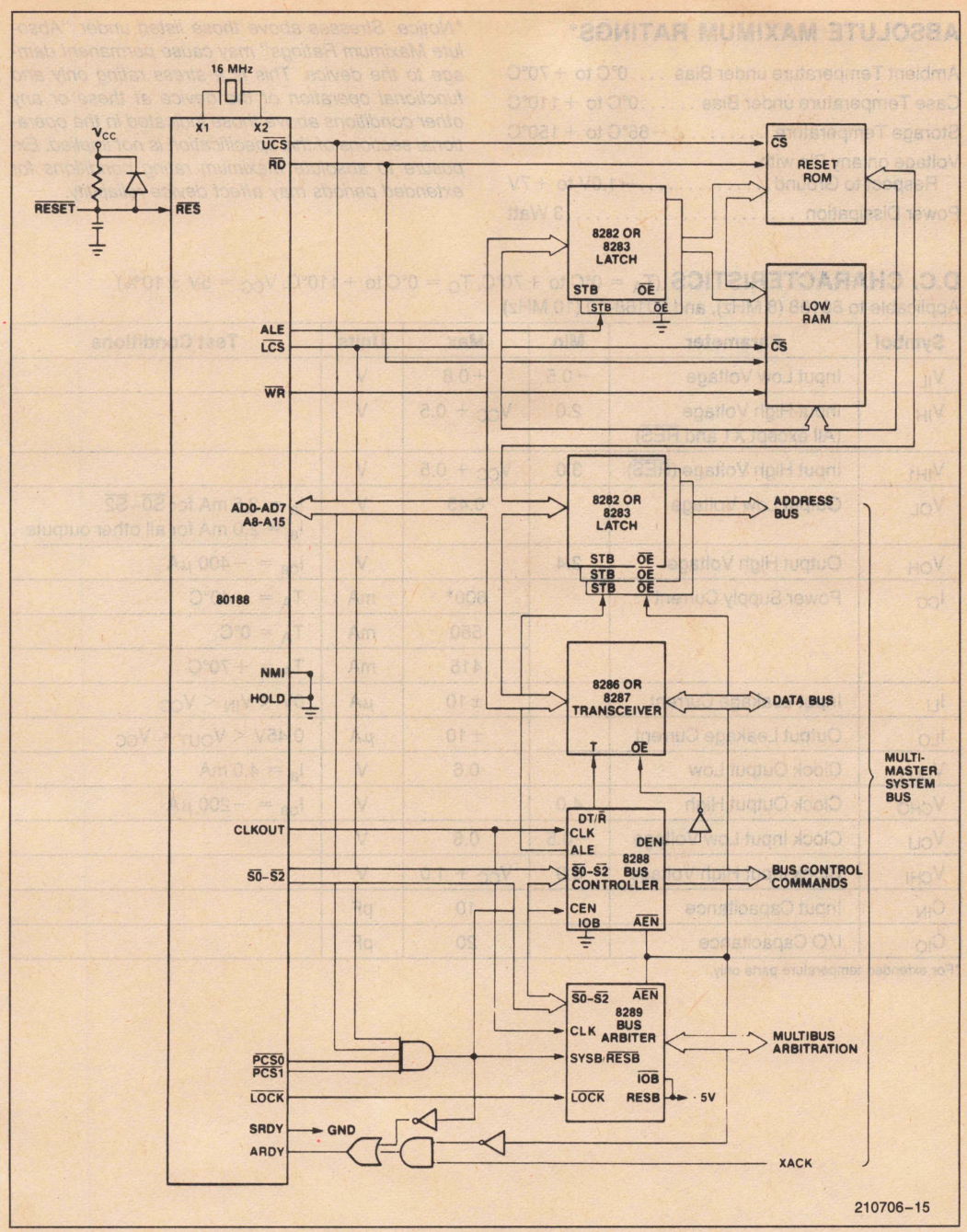


Figure 40. Typical 80188 Multi-Master Bus Interface

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature under Bias ....0°C to +70°C
Case Temperature under Bias .....0°C to +110°C
Storage Temperature .....−65°C to +150°C
Voltage on any Pin with Respect to Ground.....−1.0V to +7V
Power Dissipation .....3 Watt

Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

D.C. CHARACTERISTICS (T_A = 0°C to +70°C, T_C = 0°C to +110°C, V_{CC} = 5V ±10%)

Applicable to 80188 (8 MHz), and 80188-10 (10 MHz)

Symbol	Parameter	Min	Max	Units	Test Conditions
V_{IL}	Input Low Voltage	−0.5	+0.8	V	
V_{IH}	Input High Voltage (All except X1 and $\overline{RES}$)	2.0	V_{CC} + 0.5	V	
V_{IH1}	Input High Voltage ($\overline{RES}$)	3.0	V_{CC} + 0.5	V	
V_{OL}	Output Low Voltage		0.45	V	I_a = 2.5 mA for $\overline{S0}$–$\overline{S2}$ I_a = 2.0 mA for all other outputs
V_{OH}	Output High Voltage	2.4		V	I_{oa} = −400 μA
I_{CC}	Power Supply Current		600*	mA	T_A = −40°C
			550	mA	T_A = 0°C
			415	mA	T_A = +70°C
I_{LI}	Input Leakage Current		±10	μA	0V < V_{IN} < V_{CC}
I_{LO}	Output Leakage Current		±10	μA	0.45V < V_{OUT} < V_{CC}
V_{CLO}	Clock Output Low		0.6	V	I_a = 4.0 mA
V_{CHO}	Clock Output High	4.0		V	I_{oa} = −200 μA
V_{CLI}	Clock Input Low Voltage	−0.5	0.6	V	
V_{CHI}	Clock Input High Voltage	3.9	V_{CC} + 1.0	V	
C_{IN}	Input Capacitance		10	pF	
C_{IO}	I/O Capacitance		20	pF	

*For extended temperature parts only.

PIN TIMINGS

A.C. CHARACTERISTICS (T_A = 0°C to +70°C, T_C = 0°C to +110°C, V_{CC} = 5V ±10%)

80188 Timing Requirements All Timings Measured At 1.5 Volts Unless Otherwise Noted

Symbol	Parameter	80188 (8 MHz)		80188-10 (10 MHz)		Units	Test Conditions
		Min	Max	Min	Max		
T_{DVCL}	Data in Setup (A/D)	20		15		ns	
T_{CLDX}	Data in Hold (A/D)	10		8		ns	
T_{ARYHCH}	Asynchronous Ready (AREADY) active setup time*	20		15		ns	
T_{ARYLCL}	AREADY inactive setup time	35		25		ns	
T_{CHARYX}	AREADY hold time	15		15		ns	
T_{ARYCHL}	Asynchronous Ready inactive hold time	15		15		ns	
T_{SRYCL}	Synchronous Ready (SREADY) transition setup time	20		20		ns	
T_{CLSRY}	SREADY transition hold time	15		15		ns	
T_{HVCL}	HOLD Setup*	25		20		ns	
T_{INVCH}	INTR, NMI, $\overline{TEST}$, TIMERIN, Setup*	25		25		ns	
T_{INVCL}	DRQ0, DRQ1, Setup*	25		20		ns	

80188 Master Interface Timing Responses

T_{CLAV}	Address Valid Delay	5	55	5	44	ns	C_L = 20–200 pF all outputs (except T_{CLTMV}) @ 8 & 10 MHz
T_{CLAX}	Address Hold	10		10		ns	
T_{CLAZ}	Address Float Delay	T_{CLAX}	35	T_{CLAX}	30	ns	
T_{CHCZ}	Command Lines Float Delay		45		40	ns	
T_{CHCV}	Command Lines Valid Delay (after float)		55		45	ns	
T_{LHLL}	ALE Width	$T_{CLCL}-35$		$T_{CLCL}-30$		ns	
T_{CHLH}	ALE Active Delay		35		30	ns	
T_{CHLL}	ALE Inactive Delay		35		30	ns	
T_{LLAX}	Address Hold to ALE Inactive	$T_{CHCL}-25$		$T_{CHCL}-20$		ns	
T_{CLDV}	Data Valid Delay	10	44	10	40	ns	
T_{CLDOX}	Data Hold Time	10		10		ns	
T_{WHDX}	Data Hold after WR	$T_{CLCL}-40$		$T_{CLCL}-34$		ns	
T_{CVCTV}	Control Active Delay 1	5	50	5	40	ns	
T_{CHCTV}	Control Active Delay 2	10	55	10	44	ns	
T_{CVCTX}	Control Inactive Delay	5	55	5	44	ns	
T_{CVDEX}	$\overline{DEN}$ Inactive Delay (Non-Write Cycle)	10	70	10	56	ns	

*To guarantee recognition at next clock.

PIN TIMINGS (Continued)

A.C. CHARACTERISTICS

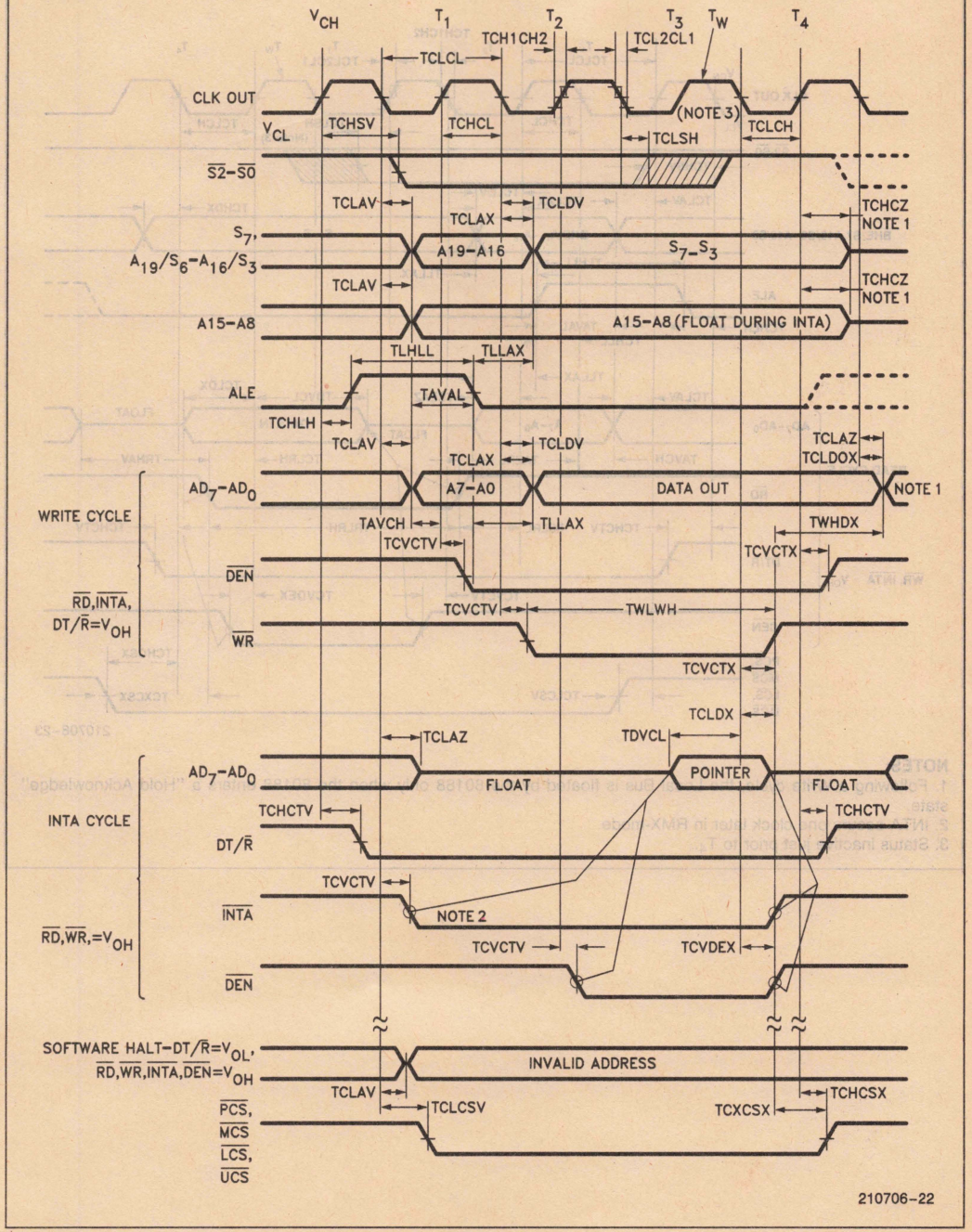
(T_A = 0°C to +70°C, T_C = 0°C to +110°C, V_{CC} = 5V ±10%) (Continued)

80188 Master Interface Timing Responses (Continued)

Symbol	Parameter	80188 (8 MHz)		80188-10 (10 MHz)		Units	Test Conditions
		Min	Max	Min	Max		
T_{AZRL}	Address Float to $\overline{RD}$ Active	0		0		ns	
T_{CLRL}	$\overline{RD}$ Active Delay	10	70	10	56	ns	
T_{CLRH}	$\overline{RD}$ Inactive Delay	10	55	10	44	ns	
T_{RHAV}	$\overline{RD}$ Inactive to Address Active	$T_{CLCL}-40$		$T_{CLCL}-40$		ns	
T_{CLHAV}	HLDA Valid Delay	5	50	5	40	ns	
T_{RLRH}	$\overline{RD}$ Width	$2T_{CLCL}-50$		$2T_{CLCL}-46$		ns	
T_{WLWH}	$\overline{WR}$ Width	$2T_{CLCL}-40$		$2T_{CLCL}-34$		ns	
T_{AVAL}	Address Valid to ALE Low	$T_{CLCH}-25$		$T_{CLCH}-19$		ns	
T_{CHSV}	Status Active Delay	10	55	10	45	ns	
T_{CLSH}	Status Inactive Delay	10	65	10	50	ns	
T_{CLTMV}	Timer Output Delay		60		48	ns	100 pF max
T_{CLRO}	Reset Delay		60		48	ns	
T_{CHQSV}	Queue Status Delay		35		28	ns	
T_{CHDX}	Status Hold Time	10		10		ns	
T_{AVCH}	Address Valid to Clock High	10		10		ns	
T_{CLLV}	$\overline{LOCK}$ Valid/Invalid Delay	5	65	5	60	ns	
80188 Chip-Select Timing Responses							
T_{CLCSV}	Chip-Select Active Delay		66		45	ns	
T_{CXCSX}	Chip-Select Hold from Command Inactive	35		35		ns	
T_{CHCSX}	Chip-Select Inactive Delay	5	35	5	32	ns	
80188 CLKIN Requirements							
T_{CKIN}	CLKIN Period	62.5	250	50	250	ns	
T_{CKHL}	CLKIN Fall Time		10		10	ns	3.5 to 1.0V
T_{CKLH}	CLKIN Rise Time		10		10	ns	1.0 to 3.5V
T_{CLCK}	CLKIN Low Time	25		20		ns	1.5V
T_{CHCK}	CLKIN High Time	25		20		ns	1.5V
80188 CLKOUT Timing (200 pF load)							
T_{CICO}	CLKIN to CLKOUT Skew		50		25	ns	
T_{CLCL}	CLKOUT Period	125	500	100	500	ns	
T_{CLCH}	CLKOUT Low Time	$\frac{1}{2}T_{CLCL}-7.5$		$\frac{1}{2}T_{CLCL}-6.0$		ns	1.5V
T_{CHCL}	CLKOUT High Time	$\frac{1}{2}T_{CLCL}-7.5$		$\frac{1}{2}T_{CLCL}-6.0$		ns	1.5V
T_{CH1CH2}	CLKOUT Rise Time		15		12	ns	1.0 to 3.5V
T_{CL2CL1}	CLKOUT Fall Time		15		12	ns	3.5 to 1.0V

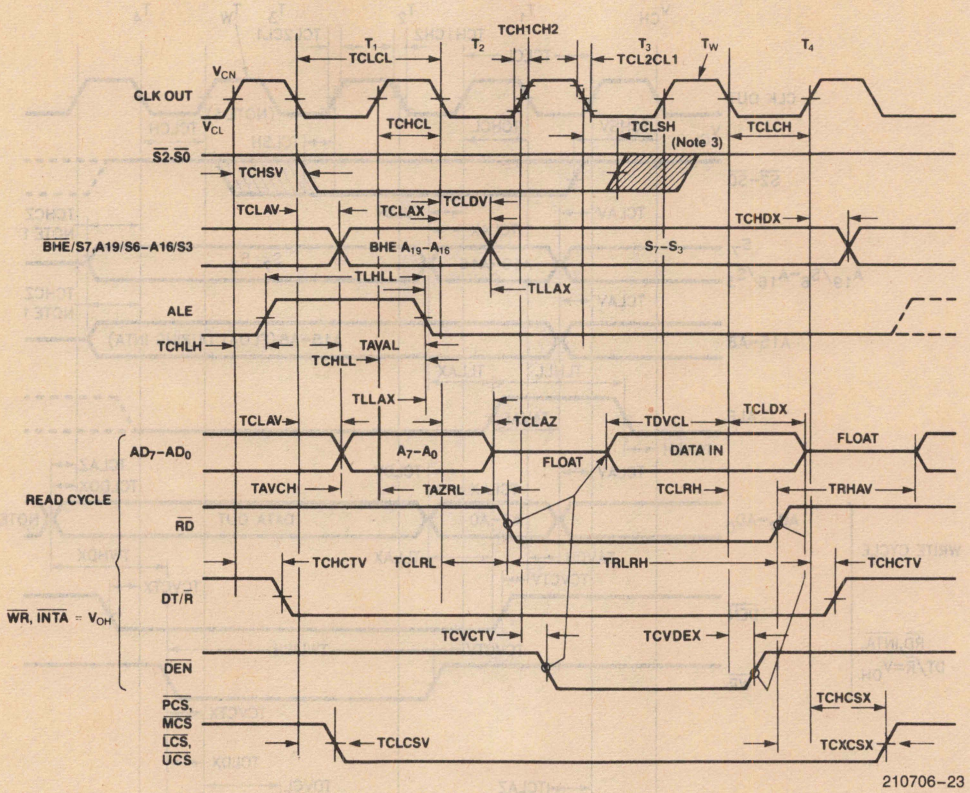
WAVEFORMS

Major Cycle Timing



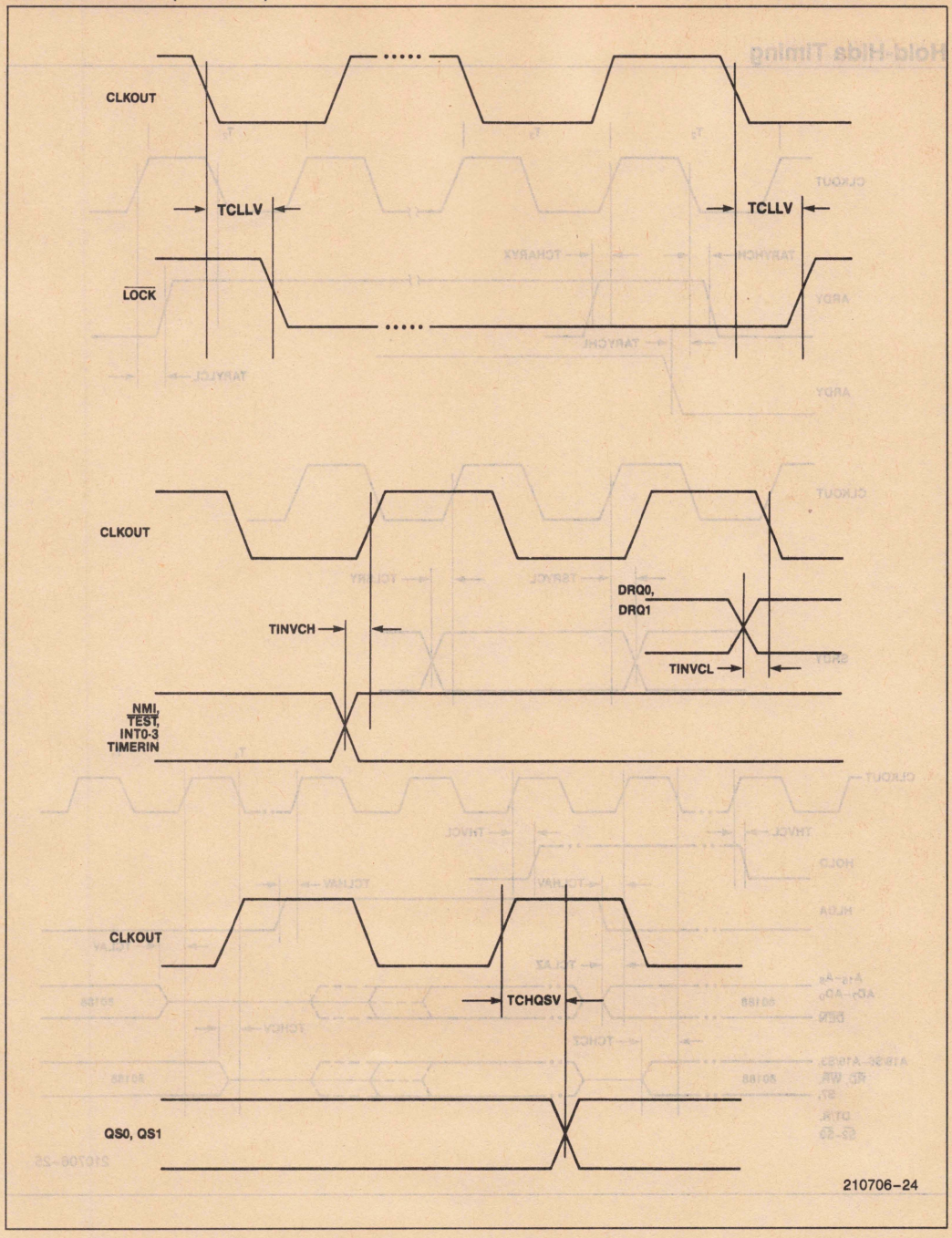
WAVEFORMS (Continued)

Major Cycle Timing (Continued)



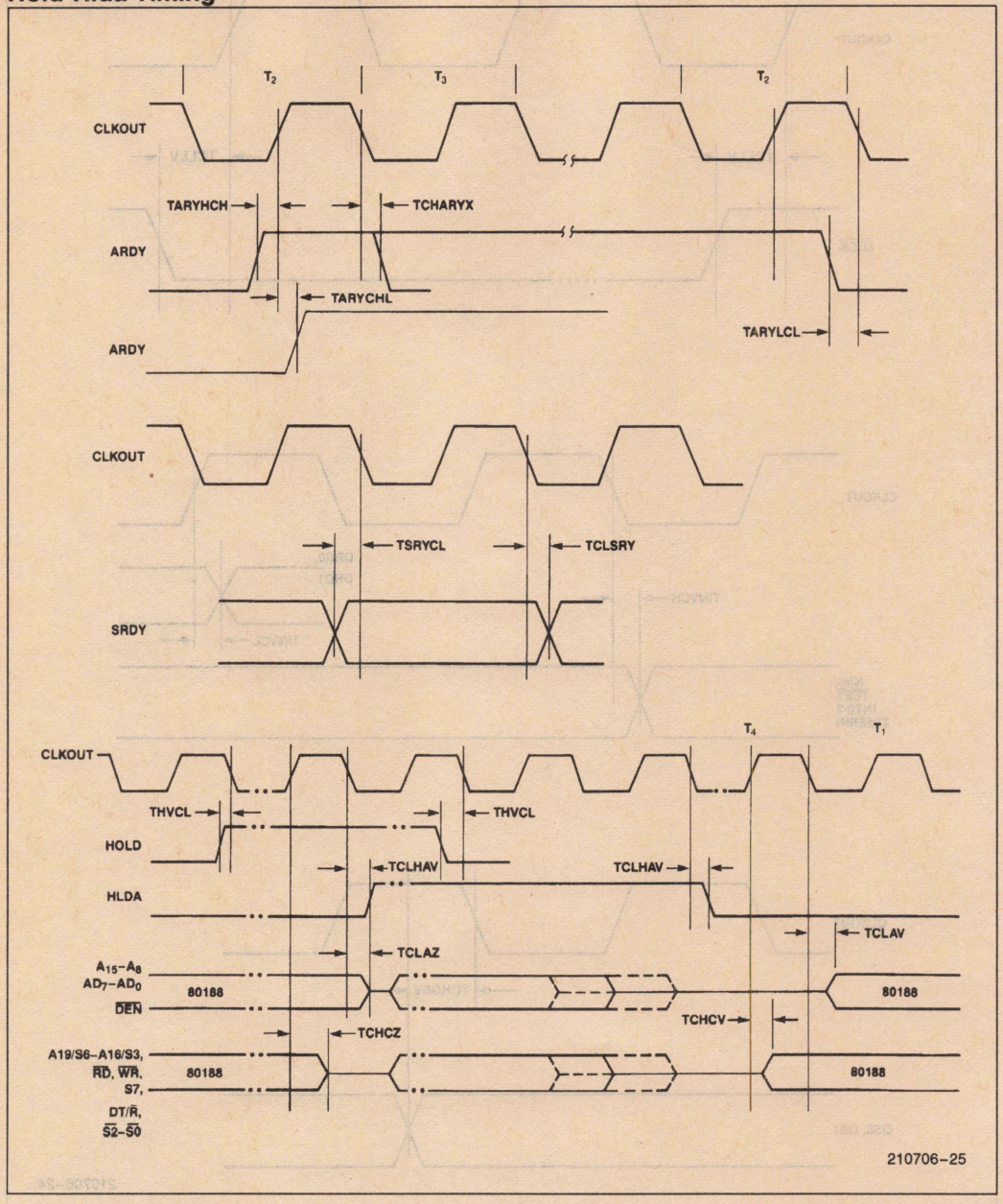
NOTES:
1. Following a Write cycle, the Local Bus is floated by the 80188 only when the 80188 enters a "Hold Acknowledge" state.
2. INTA occurs one clock later in RMX-mode.
3. Status inactive just prior to T_4.

WAVEFORMS (Continued)



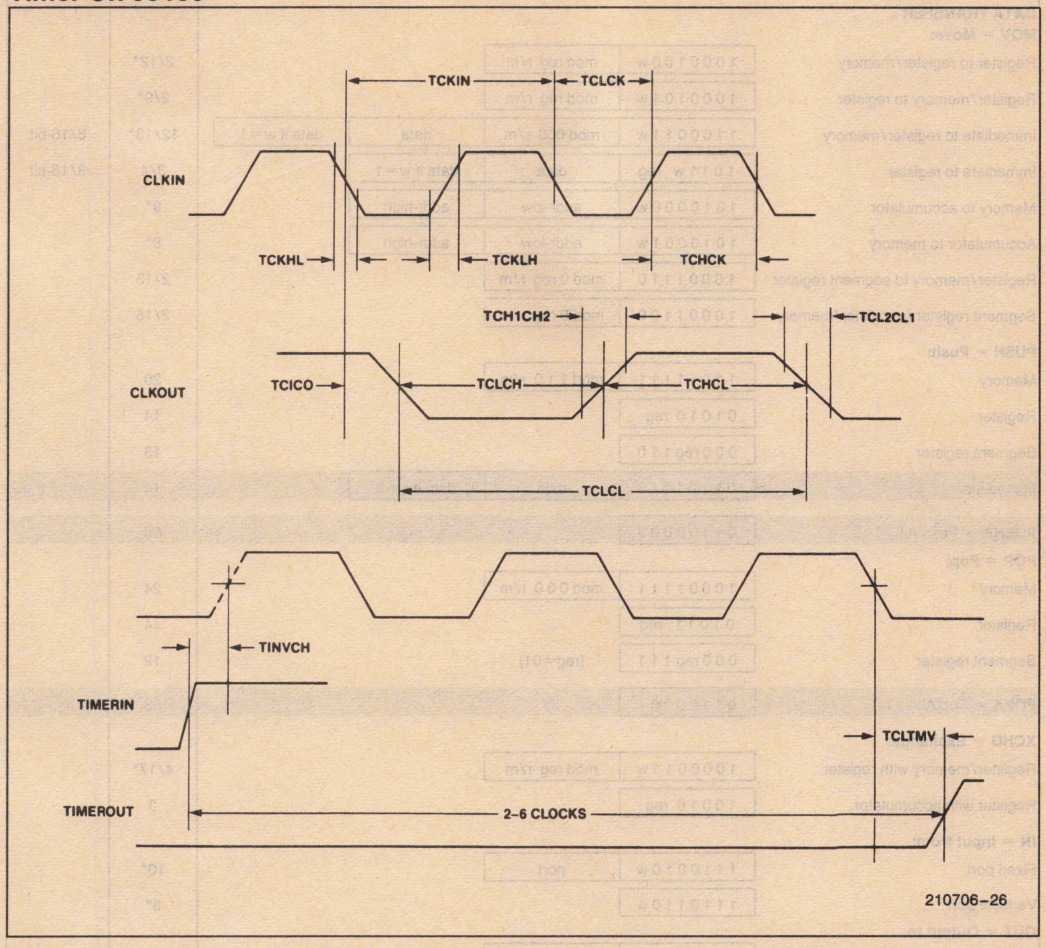
WAVEFORMS (Continued)

Hold-Hlda Timing



WAVEFORMS (Continued)

Timer On 80188



80188 INSTRUCTION TIMINGS

The following instruction timings represent the minimum execution time in clock cycles for each instruction. The timings given are based on the following assumptions:

- The opcode, along with any data or displacement required for execution of a particular instruction, has been prefetched and resides in the queue at the time it is needed.
- No wait states or bus HOLDS occur.
- All word-data is located on even-address boundaries.

All jumps and calls include the time required to fetch the opcode of the next instruction at the destination address.

All instructions which involve memory reference can require one (and in some cases, two) additional clocks above the minimum timings shown. This is due to the asynchronous nature of the handshake between the BIU and the Execution unit.

INSTRUCTION SET SUMMARY

Function	Format				Clock Cycles	Comments
DATA TRANSFER						
MOV = Move:						
Register to register/memory	1000100w	mod reg r/m			2/12*	
Register/memory to register	1000101w	mod reg r/m			2/9*	
Immediate to register/memory	1100011w	mod 000 r/m	data	data if w=1	12/13*	8/16-bit
Immediate to register	1011w reg	data	data if w=1		3/4	8/16-bit
Memory to accumulator	1010000w	addr-low	addr-high		9*	
Accumulator to memory	1010001w	addr-low	addr-high		8*	
Register/memory to segment register	10001110	mod 0 reg r/m			2/13	
Segment register to register/memory	10001100	mod 0 reg r/m			2/15	
PUSH = Push:						
Memory	11111111	mod 1 1 0 r/m			20	
Register	01010 reg				14	
Segment register	000reg110				13	
Immediate	011010s0	data	data if s=0		14	
PUSHA = Push All	01100000				68	
POP = Pop:						
Memory	10001111	mod 0 0 0 r/m			24	
Register	01011 reg				14	
Segment register	000reg111	(reg≠01)			12	
POPA = Pop All	01100001				83	
XCHG = Exchange:						
Register/memory with register	1000011w	mod reg r/m			4/17*	
Register with accumulator	10010 reg				3	
IN = Input from:						
Fixed port	1110010w	port			10*	
Variable port	1110110w				8*	
OUT = Output to:						
Fixed port	1110011w	port			9*	
Variable port	1110111w				7*	
XLAT = Translate byte to AL	11010111				15	
LEA = Load EA to register	10001101	mod reg r/m			6	
LDS = Load pointer to DS	11000101	mod reg r/m	(mod≠11)		26	
LES = Load pointer to ES	11000100	mod reg r/m	(mod≠11)		26	
LAHF = Load AH with flags	10011111				2	
SAHF = Store AH into flags	10011110				3	
PUSHF = Push flags	10011100				13	

Shaded areas indicate instructions not available in 8086, 8088 microsystems.
*Note: Clock cycles shown for byte transfer. For word operations, add 4 clock cycles for all memory transfers.

INSTRUCTION SET SUMMARY (Continued)

Function	Format				Clock Cycles	Comments
DATA TRANSFER (Continued)						
POPF = Pop flags	10011101				12	
SEGMENT = **Segment Override:**						
CS	00101110				2	
SS	00110110				2	
DS	00111110				2	
ES	00100110				2	
ARITHMETIC **ADD** = **Add:**						
Reg/memory with register to either	000000dw	mod reg r/m			3/10*	
Immediate to register/memory	100000sw	mod 000 r/m	data	data if s w=01	4/16*	
Immediate to accumulator	0000010w	data	data if w=1		3/4	8/16-bit
ADC = **Add with carry:**						
Reg/memory with register to either	000100dw	mod reg r/m			3/10*	
Immediate to register/memory	100000sw	mod 010 r/m	data	data if s w=01	4/16*	
Immediate to accumulator	0001010w	data	data if w=1		3/4	8/16-bit
INC = **Increment:**						
Register/memory	1111111w	mod 000 r/m			3/15*	
Register	01000 reg				3	
SUB = **Subtract:**						
Reg/memory and register to either	001010dw	mod reg r/m			3/10*	
Immediate from register/memory	100000sw	mod 101 r/m	data	data if s w=01	4/16*	
Immediate from accumulator	0010110w	data	data if w=1		3/4	8/16-bit
SBB = **Subtract with borrow:**						
Reg/memory and register to either	000110dw	mod reg r/m			3/10*	
Immediate from register/memory	100000sw	mod 011 r/m	data	data if s w=01	4/16*	
Immediate from accumulator	0001110w	data	data if w=1		3/4	8/16-bit
DEC = **Decrement:**						
Register/memory	1111111w	mod 001 r/m			3/15*	
Register	01001 reg				3	
CMP = **Compare:**						
Register/memory with register	0011101w	mod reg r/m			3/10*	
Register with register/memory	0011100w	mod reg r/m			3/10*	
Immediate with register/memory	100000sw	mod 111 r/m	data	data if s w=01	3/10*	
Immediate with accumulator	0011110w	data	data if w=1		3/4	8/16-bit
NEG = Change sign register/memory	1111011w	mod 011 r/m			3/10*	
AAA = ASCII adjust for add	00110111				8	
DAA = Decimal adjust for add	00100111				4	
AAS = ASCII adjust for subtract	00111111				7	
DAS = Decimal adjust for subtract	00101111				4	

Shaded areas indicate instructions not available in 8086, 8088 microsystems.
*Note: Clock cycles shown for byte transfer. For word operations, add 4 clock cycles for all memory transfers.

INSTRUCTION SET SUMMARY (Continued)

Function	Format	Clock Cycles	Comments
ARITHMETIC (Continued)			
MUL = Multiply (unsigned):	1111011w | mod 100 r/m		
Register-Byte		26–28	
Register-Word		35–37	
Memory-Byte		32–34	
Memory-Word		41–43*	
IMUL = Integer multiply (signed):	1111011w | mod 101 r/m		
Register-Byte		25–28	
Register-Word		34–37	
Memory-Byte		31–34	
Memory-Word		40–43*	
IMUL = Integer Immediate multiply (signed)	011010s1 | mod reg r/m | data | data if s=0	22–25/ 29–32	
DIV = Divide (unsigned):	1111011w | mod 110 r/m		
Register-Byte		29	
Register-Word		38	
Memory-Byte		35	
Memory-Word		44*	
IDIV = Integer divide (signed):	1111011w | mod 111 r/m		
Register-Byte		44–52	
Register-Word		53–61	
Memory-Byte		50–58	
Memory-Word		59–67*	
AAM = ASCII adjust for multiply	11010100 | 00001010	19	
AAD = ASCII adjust for divide	11010101 | 00001010	15	
CBW = Convert byte to word	10011000	2	
CWD = Convert word to double word	10011001	4	
LOGIC			
Shift/Rotate Instructions:			
Register/Memory by 1	1101000w | mod TTT r/m	2/15	
Register/Memory by CL	1101001w | mod TTT r/m	5+n/17+n	
Register/Memory by Count	1100000w | mod TTT r/m | count	5+n/17+n	
	TTT Instruction 000 ROL 001 ROR 010 RCL 011 RCR 100 SHL/SAL 101 SHR 111 SAR		
AND = And:			
Reg/memory and register to either	001000dw | mod reg r/m	3/10*	
Immediate to register/memory	1000000w | mod 100 r/m | data | data if w=1	4/16*	
Immediate to accumulator	0010010w | data | data if w=1	3/4	8/16-bit

Shaded areas indicate instructions not available in 8086, 8088 microsystems.
*Note: Clock cycles shown for byte transfer. For word operations, add 4 clock cycles for all memory transfers.

INSTRUCTION SET SUMMARY (Continued)

Function	Format				Clock Cycles	Comments
LOGIC (Continued)						
TEST = And function to flags, no result:						
Register/memory and register	1000010w	mod reg r/m			3/10*	
Immediate data and register/memory	1111011w	mod 000 r/m	data	data if w=1	4/10*	
Immediate data and accumulator	1010100w	data	data if w=1		3/4	8/16-bit
OR = Or:						
Reg/memory and register to either	000010dw	mod reg r/m			3/10*	
Immediate to register/memory	1000000w	mod 001 r/m	data	data if w=1	4/16*	
Immediate to accumulator	0000110w	data	data if w=1		3/4	8/16-bit
XOR = Exclusive or:						
Reg/memory and register to either	001100dw	mod reg r/m			3/10*	
Immediate to register/memory	1000000w	mod 110 r/m	data	data if w=1	4/16*	
Immediate to accumulator	0011010w	data	data if w=1		3/4	8/16-bit
NOT = Invert register/memory	1111011w	mod 010 r/m			3/10*	
STRING MANIPULATION:						
MOVS = Move byte/word	1010010w				14*	
CMPS = Compare byte/word	1010011w				22*	
SCAS = Scan byte/word	1010111w				15*	
LODS = Load byte/wd to AL/AX	1010110w				12*	
STOS = Stor byte/wd from AL/A	1010101w				10*	
INS = Input byte/wd from DX port	0110110w				14	
OUTS = Output byte/wd to DX port	0110111w				14	
Repeated by count in CX						
MOVS = Move string	11110010	1010010w			8+8n*	
CMPS = Compare string	1111001z	1010011w			5+22n*	
SCAS = Scan string	1111001z	1010111w			5+15n*	
LODS = Load string	11110010	1010110w			6+11n*	
STOS = Store string	11110010	1010101w			6+9n*	
INS = Input string	11110010	0110110w			8+8n*	
OUTS = Output string	11110010	0110111w			8+8n*	

Shaded areas indicate instructions not available in 8086, 8088 microsystems.
*Note: Clock cycles shown for byte transfer. For word operations, add 4 clock cycles for all memory transfers.

INSTRUCTION SET SUMMARY (Continued)

Function	Format	Clock Cycles	Comments
CONTROL TRANSFER			
CALL = Call:			
Direct within segment	11101000 | disp-low | disp-high	19	
Register/memory indirect within segment	11111111 | mod 010 r/m	17/27	
Direct intersegment	10011010 | segment offset segment selector	31	
Indirect intersegment	11111111 | mod 011 r/m | (mod ≠ 11)	54	
JMP = Unconditional jump:			
Short/long	11101011 | disp-low	14	
Direct within segment	11101001 | disp-low | disp-high	14	
Register/memory indirect within segment	11111111 | mod 100 r/m	11/21	
Direct intersegment	11101010 | segment offset segment selector	14	
Indirect intersegment	11111111 | mod 101 r/m | (mod ≠ 11)	34	
RET = Return from CALL:			
Within segment	11000011	20	
Within seg adding immed to SP	11000010 | data-low | data-high	22	
Intersegment	11001011	30	
Intersegment adding immediate to SP	11001010 | data-low | data-high	33	
JE/JZ = Jump on equal/zero	01110100 | disp	4/13	JMP not taken/JMP taken
JL/JNGE = Jump on less/not greater or equal	01111100 | disp	4/13	
JLE/JNG = Jump on less or equal/not greater	01111110 | disp	4/13	
JB/JNAE = Jump on below/not above or equal	01110010 | disp	4/13	
JBE/JNA = Jump on below or equal/not above	01110110 | disp	4/13	
JP/JPE = Jump on parity/parity even	01111010 | disp	4/13	
JO = Jump on overflow	01110000 | disp	4/13	
JS = Jump on sign	01111000 | disp	4/13	
JNE/JNZ = Jump on not equal/not zero	01110101 | disp	4/13	
JNL/JGE = Jump on not less/greater or equal	01111101 | disp	4/13	
JNLE/JG = Jump on not less or equal/greater	01111111 | disp	4/13	
JNB/JAE = Jump on not below/above or equal	01110011 | disp	4/13	
JNBE/JA = Jump on not below or equal/above	01110111 | disp	4/13	
JNP/JPO = Jump on not par/par odd	01111011 | disp	4/13	

Shaded areas indicate instructions not available in 8086, 8088 microsystems.
*Note: Clock cycles shown for byte transfer. For word operations, add 4 clock cycles for all memory transfers.

INSTRUCTION SET SUMMARY (Continued)

Function	Format				Clock Cycles	Comments
CONTROL TRANSFER (Continued)						
JNO = Jump on not overflow	01110001	disp			4/13	
JNS = Jump on not sign	01111001	disp			4/13	
JCXZ = Jump on CX zero	11100011	disp			5/15	
LOOP = Loop CX times	11100010	disp			6/16	LOOP not taken/LOOP taken
LOOPZ/LOOPE = Loop while zero/equal	11100001	disp			6/16	
LOOPNZ/LOOPNE = Loop while not zero/equal	11100000	disp			6/16	
ENTER = Enter Procedure	11001000	data-low	data-high	L		
L = 0					15	
L = 1					25	
L > 1					22+16(n−1)	
LEAVE = Leave Procedure	11001001				8	
INT = **Interrupt:**						
Type specified	11001101	type			47	
Type 3	11001100				45	if INT. taken/ if INT. not taken
INTO = Interrupt on overflow	11001110				48/4	
IRET = Interrupt return	11001111				28	
BOUND = Detect value out of range	01100010	mod reg r/m			33–35	
PROCESSOR CONTROL						
CLC = Clear carry	11111000				2	
CMC = Complement carry	11110101				2	
STC = Set carry	11111001				2	
CLD = Clear direction	11111100				2	
STD = Set direction	11111101				2	
CLI = Clear interrupt	11111010				2	
STI = Set interrupt	11111011				2	
HLT = Halt	11110100				2	
WAIT = Wait	10011011				6	if $\overline{\text{test}}$ = 0
LOCK = Bus lock prefix	11110000				2	
ESC = Processor Extension Escape	11011TTT	mod LLL r/m			6	
	(TTT LLL are opcode to processor extension)					

Shaded areas indicate instructions not available in 8086, 8088 microsystems.
*Note: Clock cycles shown for byte transfer. For word operations, add 4 clock cycles for all memory transfers.

FOOTNOTES

The Effective Address (EA) of the memory operand is computed according to the mod and r/m fields:

if mod = 11 then r/m is treated as a REG field

if mod = 00 then DISP = 0*, disp-low and disp-high are absent

if mod = 01 then DISP = disp-low sign-extended to 16-bits, disp-high is absent

if mod = 10 then DISP = disp-high: disp-low

if r/m = 000 then EA = (BX) + (SI) + DISP

if r/m = 001 then EA = (BX) + (DI) + DISP

if r/m = 010 then EA = (BP) + (SI) + DISP

if r/m = 011 then EA = (BP) + (DI) + DISP

if r/m = 100 then EA = (SI) + DISP

if r/m = 101 then EA = (DI) + DISP

if r/m = 110 then EA = (BP) + DISP*

if r/m = 111 then EA = (BX) + DISP

DISP follows 2nd byte of instruction (before data if required)

*except if mod = 00 and r/m = 110 then EA = disp-high: disp-low.

EA calculation time is 4 clock cycles for all modes, and is included in the execution times given whenever appropriate.

Segment Override Prefix

0	0	1	reg	1	1	0

reg is assigned according to the following:

reg	Segment Register
00	ES
01	CS
10	SS
11	DS

REG is assigned according to the following table:

16-Bit (w = 1)	8-Bit (w = 0)
000 AX	000 AL
001 CX	001 CL
010 DX	010 DL
011 BX	011 BL
100 SP	100 AH
101 BP	101 CH
110 SI	110 DH
111 DI	111 BH

The physical addresses of all operands addressed by the BP register are computed using the SS segment register. The physical addresses of the destination operands of the string primitive operations (those addressed by the DI register) are computed using the ES segment, which may not be overridden.

82188
INTEGRATED BUS CONTROLLER FOR 8086, 8088, 80186, 80188 PROCESSORS

- **Provides Flexibility in System Configurations**
  - **— Supports 8087 Numerics Coprocessor in 8 MHz 80186 and 80188 Systems**
  - **— Provides a Low-cost Interface for 8086, 8088 Systems to an 82586 LAN Coprocessor or 82730 Text Coprocessor**
- **Facilitates Interface to one or more Multimaster Busses**
- **Supports Multiprocessor, Local Bus Systems**
- **Allows use of 80186, 80188 High-Integration Features**
- **3-State, Command Output Drivers**
- **Available in EXPRESS**
  - **— Standard Temperature Range**
  - **— Extended Temperature Range**
- **Available in Plastic DIP or Cerdip Package**

(See Packaging Spec., Order #231369)

The 82188 Integrated Bus Controller (IBC) is a 28-pin HMOS III component for use with 80186, 80188, 8086 and 8088 systems. The IBC provides command and control timing signals plus a configurable $\overline{RQ}/\overline{GT} \longleftrightarrow$ HOLD-HLDA converter. The device may be used to interface an 8087 Numerics Coprocessor with an 80186 or 80188 Processor. Also, an 82586 Local Area Network (LAN) Coprocessor or 82730 Text Coprocessor may be interfaced to an 8086 or 8088 with the IBC.

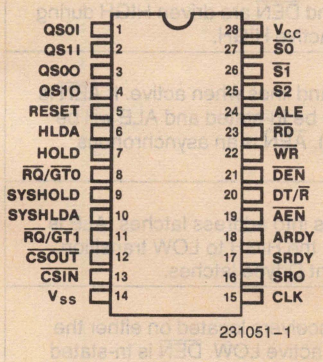


Figure 1.
82188 Pin Configuration

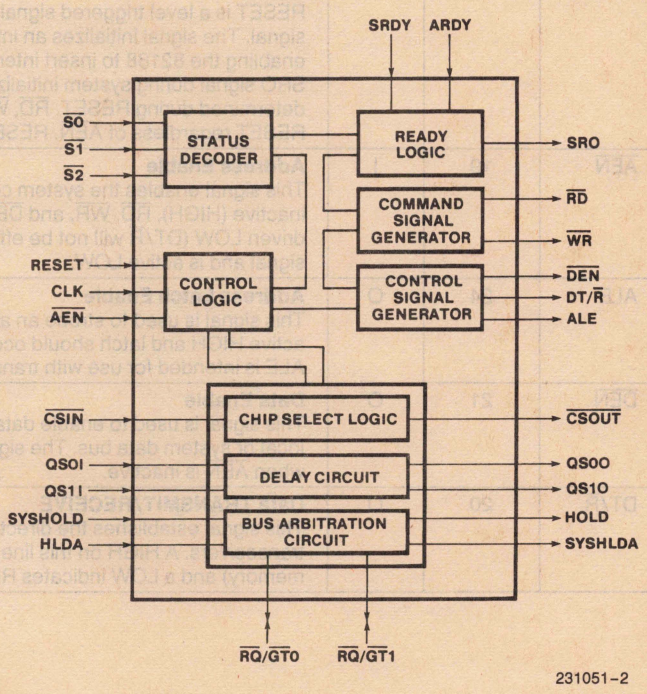


Figure 2.
82188 Block Diagram

October 1986
Order Number: 231051-004

PIN DESCRIPTIONS

<table>
<tr><th>Symbol</th><th>Pin No.</th><th>Type</th><th>Name and Function</th></tr>
<tr><td>$\overline{S0}$
$\overline{S1}$
$\overline{S2}$</td><td>27
26
25</td><td>I</td><td>Status Input Pins
$\overline{S0}$–$\overline{S2}$ correspond to the status pins of the CPU. The 82188 uses the status lines to detect and identify the processor bus cycles. The 82188 decodes $\overline{S0}$–$\overline{S2}$ to generate the command and control signals. $\overline{S0}$–$\overline{S2}$ are also used to insert 3 wait states into the SRO line during the first 256 80186 bus cycles after RESET. A HIGH input on all three lines indicates that no bus activity is taking place. The status input lines contain weak internal pull-up devices.
<table>
<tr><th>$\overline{S2}$</th><th>$\overline{S1}$</th><th>$\overline{S0}$</th><th>Bus Cycle Initiated</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>interrupt acknowledge</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>read I/O</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>write I/O</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>halt</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>instruction fetch</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>read data from memory</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>write data to memory</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>passive (no bus cycle)</td></tr>
</table></td></tr>
<tr><td>CLK</td><td>15</td><td>I</td><td>CLOCK
CLK is the clock signal generated by the CPU or clock generator device. CLK edges establish when signals are sampled and generated.</td></tr>
<tr><td>RESET</td><td>5</td><td>I</td><td>RESET
RESET is a level triggered signal that corresponds to the system reset signal. The signal initializes an internal bus cycle counter, thus enabling the 82188 to insert internally generated wait states into the SRO signal during system initialization. The 82188 mode is also determined during RESET. $\overline{RD}$, $\overline{WR}$, and $\overline{DEN}$ are driven HIGH during RESET regardless of $\overline{AEN}$. RESET is active HIGH.</td></tr>
<tr><td>$\overline{AEN}$</td><td>19</td><td>I</td><td>Address Enable
This signal enables the system command lines when active. If $\overline{AEN}$ is inactive (HIGH), $\overline{RD}$, $\overline{WR}$, and $\overline{DEN}$ will be tri-stated and ALE will be driven LOW (DT/$\overline{R}$ will not be effected). $\overline{AEN}$ is an asynchronous signal and is active LOW.</td></tr>
<tr><td>ALE</td><td>24</td><td>O</td><td>Address Latch Enable
This signal is used to strobe an address into address latches. ALE is active HIGH and latch should occur on the HIGH to LOW transition. ALE is intended for use with transparent D-type latches.</td></tr>
<tr><td>$\overline{DEN}$</td><td>21</td><td>O</td><td>Data Enable
This signal is used to enable data transceivers located on either the local or system data bus. The signal is active LOW. $\overline{DEN}$ is tri-stated when $\overline{AEN}$ is inactive.</td></tr>
<tr><td>DT/$\overline{R}$</td><td>20</td><td>O</td><td>Data TRANSMIT/RECEIVE
This signal establishes the direction of data flow through the data transceivers. A HIGH on this line indicates TRANSMIT (write to I/O or memory) and a LOW indicates RECEIVE (Read from I/O or memory).</td></tr>
</table>

PIN DESCRIPTIONS (Continued)

Symbol	Pin No.	Type	Name and Function
$\overline{RD}$	23	O	**READ** This signal instructs an I/O or memory device to drive its data onto the data bus. The $\overline{RD}$ signal is similiar to the $\overline{RD}$ signal of the 80186(80188) in Non-Queue-Status Mode. $\overline{RD}$ is active LOW and is tri-stated when $\overline{AEN}$ is inactive.
$\overline{WR}$	22	O	**WRITE** This signal instructs an I/O or memory device to record the data presented on the data bus. The $\overline{WR}$ signal is similiar to the $\overline{WR}$ signal of the 80186(80188) in Non-Queue-Status Mode. $\overline{WR}$ is active LOW and is tri-stated when $\overline{AEN}$ is inactive.
HOLD	7	O	**HOLD** The HOLD signal is used to request bus control from the 80186 or 80188. The request can come from either the 8087 ($\overline{RQ}/\overline{GT}O$) or from the third processor (SYSHOLD). The signal is active HIGH.
HLDA	6	I	**HOLD Acknowledge** 80186 MODE–This line serves to translate the HLDA output of the 80186(80188) to the appropriate signal of the device requesting the bus. HLDA going active (HIGH) indicates that the 80186 has relinquished the bus. If the requesting device is the 8087, HLDA will be translated into the grant pulse of the $\overline{RQ}/\overline{GT}O$ line. If the requesting device is the optional third processor, HLDA will be routed into the SYSHLDA line. This pin also determines the mode in which the 82188 will operate. If this line is HIGH during the falling edge of RESET, the 82188 will enter the 8086 mode. If LOW, the 82188 will enter the 80186 mode. For 8086 mode, this pin should be strapped to V_{CC}.
$\overline{RQ}/\overline{GT}O$	8	I/O	**Request/Grant O** $\overline{RQ}/\overline{GT}O$ is connected to $\overline{RQ}/\overline{GT}O$ of the 8087 Numeric Coprocessor. When initiated by the 8087, $\overline{RQ}/\overline{GT}O$ will be translated to HOLD-HLDA to acquire the bus from the 80186(80188). This line is bidirectional, and is active LOW. $\overline{RQ}/\overline{GT}O$ has a weak internal pull-up device to prevent erroneous request/grant signals.
$\overline{RQ}/\overline{GT}1$	11	I/O	**Request/Grant 1** 80186 Mode–In 80186 Mode, $\overline{RQ}/\overline{GT}1$ allows a third processor to take control of the local bus when the 8087 has bus control. For a HOLD-HLDA type third processor, the 82188's $\overline{RQ}/\overline{GT}1$ line should be connected to the $\overline{RQ}/\overline{GT}1$ line of the 8087. 8086 MODE–In 8086 Mode, $\overline{RQ}/\overline{GT}1$ is connected to either $\overline{RQ}/\overline{GT}O$ or $\overline{RQ}/\overline{GT}1$ of the 8086. $\overline{RQ}/\overline{GT}1$ will start its request/grant sequence when the SYSHOLD line goes active. In 8086 Mode, $\overline{RQ}/\overline{GT}1$ is used to gain bus control from the 8086 or 8088. $\overline{RQ}/\overline{GT}1$ is a bidirectional line and is active LOW. This line has a weak internal pull-up device to prevent erroneous request/grant signals.

PIN DESCRIPTIONS (Continued)

Symbol	Pin No.	Type	Name and Function
SYSHOLD	9	I	**System Hold** 80186 MODE–SYSHOLD serves as a hold input for an optional third processor in an 80186(80188)-8087 system. If the 80186(80188) has bus control, SYSHOLD will be routed to HOLD to gain control of the bus. If the 8087 has bus control, SYSHOLD will be translated to $\overline{RQ}/\overline{GT}1$ to gain control of the bus. 8086 MODE–SYSHOLD serves as a hold input for a coprocessor in an 8086 or 8088 system. SYSHOLD is translated to $\overline{RQ}/\overline{GT}1$ of the 82188 to allow the coprocessor to take control of the bus. SYSHOLD may be an asynchronous signal.
SYSHLDA	10	O	**System Hold Acknowledge** SYSHLDA serves as a hold acknowledge line to the processor or coprocessor connected to it. The device connected to the SYSHOLD-SYSHLDA lines is allowed the bus when SYSHLDA goes active (HIGH).
SRDY	17	I	**Synchronous Ready** The SRDY input serves the same function as SRDY of the 80186(80188). The 82188 combines SRDY with ARDY to form a synchronized ready output signal (SRO). SRDY must be synchronized external to the 82188 and is active HIGH. If tied to V_{CC}, SRO will remain active (HIGH) after the first 256 80186 cycles following RESET. If only ARDY is to be used, SRDY should be tied LOW.
ARDY	18	I	**Asynchronous Ready** The ARDY input serves the same function as ARDY of the 80186(80188). ARDY may be an asynchronous input, and is active HIGH. Only the rising edge of ARDY is synchronized by the 82188. The falling edge must be synchronized external to the 82188. If connected to V_{CC}, SRO will remain active (HIGH) after the first 256 80186 bus cycles following RESET. If only SRDY is to be used, ARDY should be connected LOW.
SRO	16	O	**Synchronous READY Output** SRO provides a synchronized READY signal which may be interfaced directly with the SRDY of the 80186(80188) and READY of the 8087. The SRO signal is an accumulation of the synchronized ARDY signal, the SRDY signal, and the internally generated wait state signal.
QS0I QS1I	1 2	I	**Queue-Status Inputs** QS0I, QS1I are connected to the Queue-Status lines of the 80186(80188) to allow synchronization of the queue-status signals to 8087 timing requirements.
QS0O QS1O	3 4	O	**Queue-Status Outputs** QS0O, QS1O are connected to the queue-status pins of the 8087. The signals produced meet 8087 Queue-Status input requirements.

PIN DESCRIPTIONS (Continued)

Symbol	Pin No.	Type	Name and Function
$\overline{CSIN}$	13	I	**Chip-Select Input** $\overline{CSIN}$ is connected to one of the chip-select lines of the 80186(80188). $\overline{CSIN}$ informs the 82188 that a bank select is taking place. The 82188 routes this signal to the chip-select output ($\overline{CSOUT}$). $\overline{CSIN}$ is active LOW. This line is not used when memory and I/O device addresses are decoded external to the 80186(80188).
$\overline{CSOUT}$	12	O	**Chip-Select Output** This signal is used as a chip-select line for a bank of memory devices. It is active when $\overline{CSIN}$ is active or when the 8087 has bus control. $\overline{CSOUT}$ is active LOW.

FUNCTIONAL DESCRIPTION

BUS CONTROLLER

The 82188 Integrated Bus Controller (IBC) generates system control and command signals. The signals generated are determined by the Status Decoding Logic. The bus controller logic interprets status lines $\overline{S0}$–$\overline{S2}$ to determine what type of bus cycle is taking place. The appropriate signals are then generated by the Command and Control Signal Generators.

The Address Enable ($\overline{AEN}$) line allows the command and control signals to be disabled. When $\overline{AEN}$ is inactive (HIGH), the command signals and $\overline{DEN}$ will be tri-stated, and ALE will be held low (DT/$\overline{R}$ will be uneffected). $\overline{AEN}$ inactive will allow other systems to take control of the bus. Control and command signals respond to a change in the $\overline{AEN}$ signal within 40 ns.

The command signals consist of $\overline{RD}$ and $\overline{WR}$. The 82188's $\overline{RD}$ and $\overline{WR}$ signals are similiar to $\overline{RD}$ and $\overline{WR}$ of the 80186(80188) in the non-Queue-Status Mode. These command signals do not differentiate between memory and I/O devices. $\overline{RD}$ and $\overline{WR}$ can be conditioned by $\overline{S2}$ of the 80186(80188) to obtain separate signals for I/O and memory devices.

The control commands consist of Data Enable ($\overline{DEN}$), Data Transmit/Receive (DT/$\overline{R}$), and Address Latch Enable (ALE). The control commands are similiar to those generated by the 80186(80188). $\overline{DEN}$ determines when the external bus should be enabled onto the local bus. DT/$\overline{R}$ determines the direction of the data transfer, and ALE determines when the address should be strobed into the latches (used for demultiplexing the address bus).

MODE SELECT

The 82188 Integrated Bus Controller (IBC) is configurable. The device has two modes: 80186 Mode and 8086 Mode. Selecting the mode of the device configures the Bus Arbitration Logic (see BUS ARBITRATION section for details). In 80186 Mode, the 82188 IBC may be used as a bus controller/interface device for an 80186(81088), 8087, and optional third processor system. In 8086 Mode, the 82188 IBC may be used as an interface device allowing a maximum mode 8086(8088) to interface with a coprocessor that uses a HOLD-HLDA bus exchange protocol.

The mode of the 82188 is determined during RESET. If the HLDA line is LOW at the falling edge of RESET (as in the case when tied to the HLDA line of the 80186 or 80188), the 82188 will enter into 80186 Mode. If the HLDA line is HIGH at the falling edge of RESET, the 82188 will enter 8086 Mode. In 8086 Mode, only the Bus Arbitration Logic is used. The eight pins used in 8086 Mode are: SYSHOLD, SYSHLDA, HLDA, CLK, RESET, $\overline{RQ}/\overline{GT1}$, V_{CC}, and V_{SS}. The other pins may be left unconnected.

BUS ARBITRATION

The Bus Exchange Logic interfaces up to three sets of bus exchange signals:

- HOLD-HLDA
- SYSHOLD-SYSHLDA
- $\overline{RQ}/\overline{GT0}$ ($\overline{RQ}/\overline{GT1}$)

This logic executes translating, routing, and arbitrating functions. The logic translates HOLD-HLDA signals to $\overline{RQ}/\overline{GT}$ signals and $\overline{RQ}/\overline{GT}$ signals to HOLD-HLDA signals. The logic also determines which set of bus exchange signals are to be interfaced. The mode of the 82188 and the priority of the devices requesting the bus determine the routing of the bus exchange signals.

80186 MODE

In 80186 Mode, a system may have three potential bus masters: the 80186 or 80188 CPU, the 8087 Numerics Coprocessor, and a third processor (such as the 82586 LAN or 82730 Text Coprocessor). The third processor may have either a HOLD-HLDA or $\overline{RQ}/\overline{GT}$ bus exchange protocol. The possible bus exchange signal connections and paths for 80186 Mode are shown in Figures 3 & 4 and Tables 1 & 2, respectively. If no HOLD-HLDA type third processor is used, SYSHOLD should be tied LOW to prevent an erroneous SYSHOLD signal. In 80186 mode, the bus priorities are:

Highest Priority.....Third Processor
Second Highest Priority.....8087
Default Priority.....80186

— THREE-PROCESSOR SYSTEM OPERATION (HOLD-HLDA TYPE THIRD PROCESSOR)

In the configuration shown in Figure 3, the third processor requests the bus by sending SYSHOLD HIGH. The 82188 will route (and translate if necessary) the request to the current bus master. This includes routing the request to HOLD if the 80186(80188) is the current bus master or routing and translating the request to $\overline{RQ}/\overline{GT}1$ if the 8087 is in control of the bus. The third processor's request is not passed through the 8087 if the 80186 is the bus master (see Table 1).

The 8087 requests the bus using $\overline{RQ}/\overline{GT}0$. The request pulse from the 8087 will be translated and routed to HOLD if the 80186 is the bus master. If the third processor has control of the bus, the grant pulse to the 8087 will be delayed until the third processor relinquishes the bus (sending SYSHOLD LOW). In this case, HOLD will remain HIGH during the third processor-to-8087 bus control transfer. The 80186 will not be granted the bus until both coprocessors have released it.

Table 1. Bus Exchange Paths (80186 Mode) (HOLD-HLDA Type 3rd Proc)

Requesting Device	Current Bus Master		
	80186	8087	3rd Proc
80186	n/a	n/a	n/a
8087	$\overline{RQ}/\overline{GT}0 \longleftrightarrow \frac{HOLD}{HLDA}$	n/a	n/a
3rd Proc	$\frac{SYSHOLD}{SYSHLDA} \longleftrightarrow \frac{HOLD}{HLDA}$	$\frac{SYSHOLD}{SYSHLDA} \longleftrightarrow \overline{RQ}/\overline{GT}1$	n/a

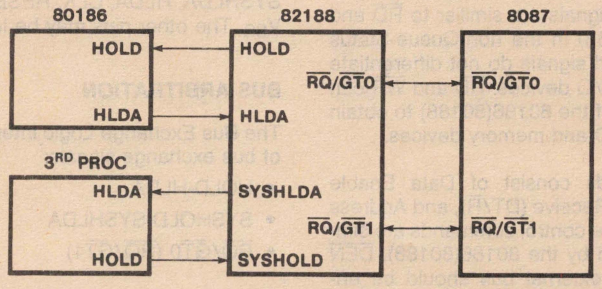


Figure 3.
Bus Exchange Signal Connections (80186 Mode) for a Three Local Processor System (HOLD-HLDA Type 3rd Proc)

Table 2. Bus Exchange Paths (80186 Mode) ($\overline{RQ}/\overline{GT}$ Type 3rd Proc)

Requesting Device	Current Bus Master		
	80186	8087	3rd Proc
80186	n/a	n/a	n/a
8087	$\overline{RQ}/\overline{GT}0 \longleftrightarrow \frac{HOLD}{HLDA}$	n/a	n/a
3rd Proc	$\overline{RQ}/\overline{GT}1 \longleftrightarrow \overline{RQ}/\overline{GT}0 \longleftrightarrow \frac{HOLD}{HLDA}$	$\overline{RQ}/\overline{GT}1$	n/a

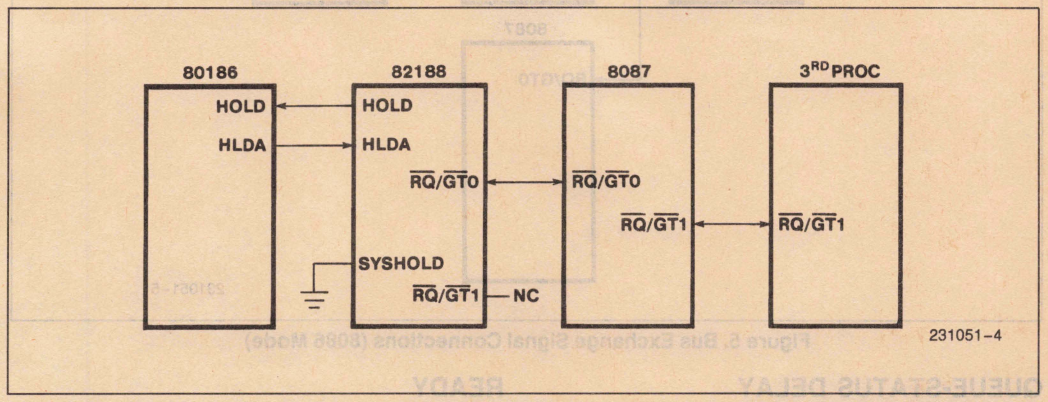


Figure 4.
Bus Exchange Signal Connections (80186 Mode) for a Three Local Processor System ($\overline{RQ}/\overline{GT}$ Type 3rd Proc)

When the bus is requested from the 80186(80188), a bus priority decision is made. This decision is made when the HLDA line goes active. Upon receipt of the HLDA signal, the highest-priority requesting device will be acknowledged the bus. For example, if the 8087 initially requested the bus, the bus will be granted to the third processor if SYSHOLD became active before HLDA was received by the 82188. In this case, the grant pulse to the 8087 will be delayed until the third processor relinquishes the bus.

— THREE-PROCESSOR SYSTEM OPERATION ($\overline{RQ}/\overline{GT}$ TYPE THIRD PROCESSOR)

In the configuration shown in Figure 4, the third processor requests the bus by initiating a request/grant sequence with the 8087's $\overline{RQ}/\overline{GT}1$ line. The 8087 will grant the bus if it is the current bus master or will pass the request on if the 80186 is the current bus master (see Table 2). In this configuration, the 82188's Bus Arbitration Logic translates $\overline{RQ}/\overline{GT}0$ to HOLD-HLDA. The 8087 provides the bus arbitration in this configuration.

8086 MODE

The 8086 Mode allows an 8086, 8088 system to contain both $\overline{RQ}/\overline{GT}$ and HOLD-HLDA type coprocessors simultaneously. In 8086 Mode, two possible bus masters may be interfaced by the 82188; an 8086 or 8088 CPU and a coprocessor which uses a HOLD-HLDA bus exchange protocol (typically an 82586 LAN Coprocessor or an 82730 Text Coprocessor). The bus exchange signal connections for 8086 Mode are shown in Figure 5. Bus arbitration signals used in the 8086 Mode are:

- $\overline{RQ}/\overline{GT}1$
- SYSHOLD
- SYSHLDA

In 8086 Mode, no arbitration is necessary since only two devices are interfaced. The coprocessor has bus priority over the 8086(8088). SYSHOLD-SYSHLDA are routed and translated directly to $\overline{RQ}/\overline{GT}1$. $\overline{RQ}/\overline{GT}1$ of the 82188 may be tied to either $\overline{RQ}/\overline{GT}0$ or $\overline{RQ}/\overline{GT}1$ of the 8086(8088).

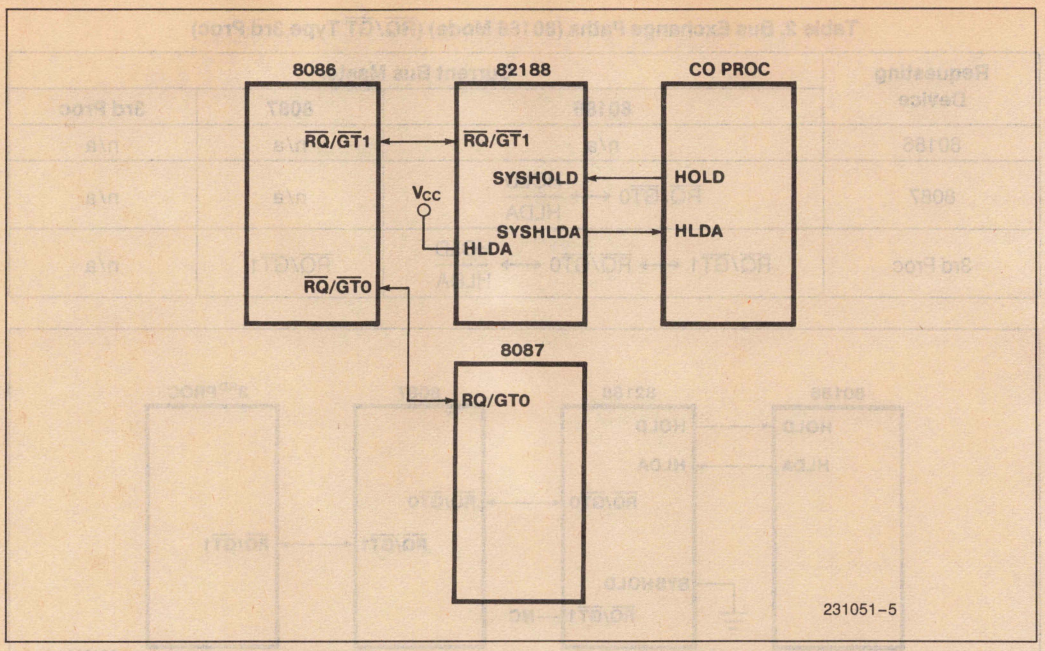


Figure 5. Bus Exchange Signal Connections (8086 Mode)

QUEUE-STATUS DELAY

The Queue-Status Delay logic is used to delay the queue-status signals from the 80186(80188) to meet 8087 queue-status timing requirements. QS0I, QS1I correspond to the queue-status lines of the 80186(80188). The 82188 delays these signals by one clock phase. The delayed signals are interfaced to the 8087 queue-status lines by QS0O, QS1O.

CHIP-SELECT

The Chip-Select Logic allows the utilization of the chip select circuitry of the 80186(80188). Normally, this circuitry could not be used in an 80186(80188)-8087 system since the 8087 contains no chip select circuitry. The Chip-Select Logic contains two external connections: Chip-Select Input ($\overline{CSIN}$) and Chip-Select Output ($\overline{CSOUT}$). $\overline{CSOUT}$ is active when either $\overline{CSIN}$ is active or when the 8087 has control of the bus.

By using $\overline{CSOUT}$ to select memory containing data structures, no external decoding is necessary. The 80186 may gain access to this memory bank through the $\overline{CSIN}$ line while the 8087 will automatically obtain access when it becomes the bus master. Note that this configuration limits the amount of memory accessible by the 8087 to the physical memory bank selected by $\overline{CSOUT}$. Systems where the 8087 must access the full 1 Megabyte address space must use an external decoding scheme.

READY

The Ready logic allows two types of Ready signals: a Synchronous Ready Signal (SRDY) and an Asynchronous Ready Signal (ARDY). These signals are similiar to SRDY and ARDY of the 80186. Wait states will be inserted when both SRDY and ARDY are LOW. Inserting wait states allows slower memory and I/O devices to be interfaced to the 80186(80188)-8087 system.

ARDY's LOW-to-HIGH transition is synchronized to the CPU clock by the 82188. The 82188 samples ARDY at the beginning of T2, T3 and Tw until sampled HIGH. Note that ARDY of the 82188 is sampled one phase earlier than ARDY of the 80186. ARDY's falling edge must be synchronous to the CPU clock. ARDY allows an easy interface with devices that emit an asynchronous ready signal.

The SRDY signal allows direct interface to devices that emit a synchronized ready signal. SRDY must be synchronized to the CPU clock for both of its transitions. SRDY is sampled in the middle of T2, T3 and in the middle of each Tw. An 82188-80186(80188)'s SRDY setup time is 30 ns longer than the 80186(80188)'s SRDY setup time. SRDY eliminates the half-clock cycle penalty necessary for ARDY to be internally sychronized.

The sychronized ready output (SRO) is the accumulation of SRDY, ARDY, and the internal wait-state

generator. SRO should be connected to SRDY of the 80186(80188) (with 80186(80188)'s ARDY tied LOW), and READY of the 8087.

SRDY	ARDY	SRO
0	0	0
1	X	1
X	1	1

The internal wait state generator allows for synchronization between the 80186(80188) and 8087 in 80186 mode. Upon RESET, the 82188 automatically inserts 3 wait-states per 80186(80188) bus cycle, overlapped with any externally produced wait-states created by ARDY and SRDY.

Since the 8087 has no provision for internal wait-state generation, only externally created wait states will be effective. The 82188, upon RESET, will inject 3 wait states for each of the first 256 80186(80188) bus cycles onto the SRO line. This will allow the 8087 to match the 80186(80188)'s timing.

The internally-generated wait states are overlapped with those produced by the SRDY and ARDY lines. Overlapping the injected wait states insures a minimum of three wait states for the first 256 80186(80188) bus cycles after RESET. Systems with a greater number of wait states will not be effected. Internal wait state generation by the 82188 will stop on the 256th 80186(80188) bus cycle after RESET. To maintain sychronization between the 80186(80188) and 8087, the following conditions are necessary:

- The 80186(80188)'s control block must be mapped in I/O space before it is written to or read from.
- All memory chip-select lines must be set to 0 WAIT STATES, EXTERNAL READY ALSO USED within the first 256 80186(80188) bus cycles after RESET.

An equivalent READY logic diagram is shown in Figure 6.

SYSTEM CONSIDERATIONS

In any 82188 configuration, clock compatibility must be considered. Depending on the device, a 50% or a 33% duty-cycle clock is needed. For example, the 80186 and 80188 (as well as the 82188, 82586, and 82730) requires a 50% duty-cycle clock. The 8086, 8088 and their 'kit' devices' (8087, 8089, 8288, and 8289) clock requirements, on the other hand, require a 33% duty-cycle clock signal. The system designer must make sure clock requirements of all the devices in the system are met.

Figure 7 demonstrates the usage of the 82188 in 80186 Mode where it is used to interface an 8087 into an 80186 system.

Status bit six (S6) from the main processor (8086, 8088, 80186, or 80188) is used by the 8087 to track the instruction flow. S6 is multiplexed with address bit 19 (A19). If the third processor generates only 16 bits of address, S6 is not generated. A19/S6 must be driven high by external circuitry during the status portion of bus cycles controlled by the third processor.

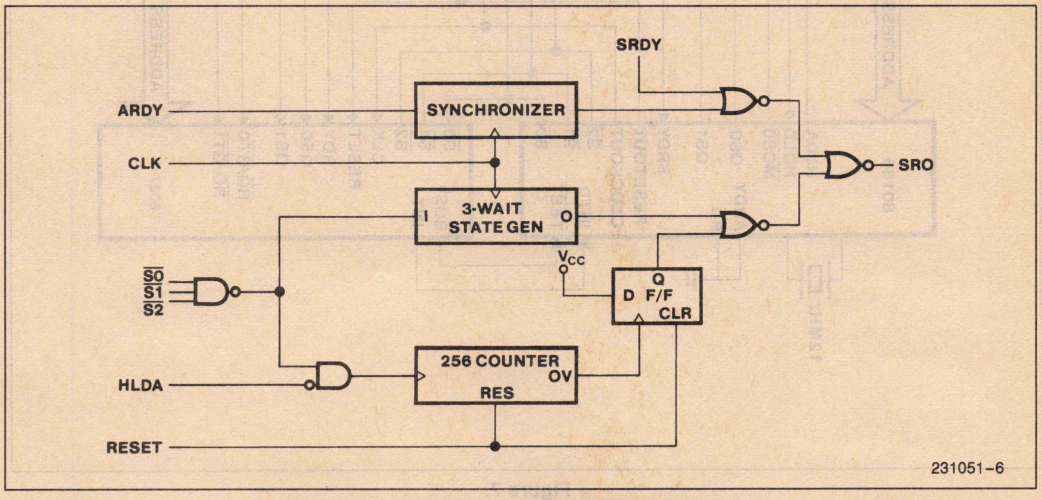


Figure 6. Equivalent 82188 READY Circuit

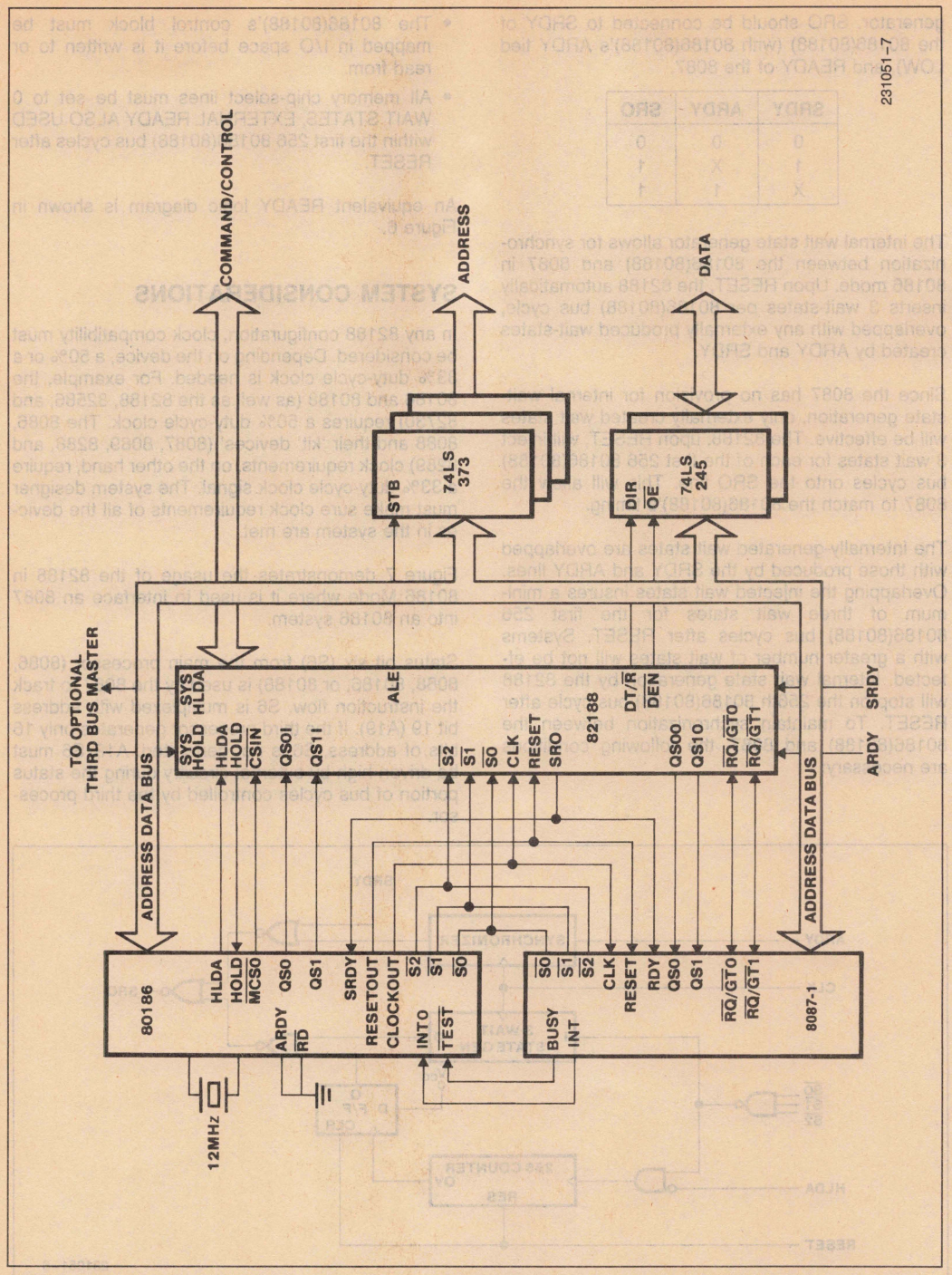


Figure 7.
80186-6/8087-2 System Using the 82188 in 80186 Mode

ABSOLUTE MAXIMUM RATINGS *

Temperature Under Bias .....0°C to 70°C
Storage Temperature.....−65°C to 150°C
Case Temperature .....0°C to +85°C
Voltage on any Pin with Respect to GND .....−1.0V to 7.0V
Power Dissipation.....0.7 Watts

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

NOTICE: Specifications contained within the following tables are subject to change.

DC CHARACTERISTICS

(V_{CC} = 5V ± 10%, T_A = 0°C to 70°C, T_{CASE} = 0°C to +85°C)

Symbol	Parameter	Min	Max	Units	Test Cond.
V_{IL}	Input Low Voltage	−0.5	+0.8	volts	
V_{IH}	Input High Voltage	2.0	V_{CC} + 0.5	volts	
V_{OL}	Output Low Voltage		0.45	volts	I_{OL} = 2 mA
V_{OH}	Output High Voltage	2.4		volts	I_{OH} = −400 μA
I_{CC}	Power Supply Current		100	mA	T_A = 25°C
I_{LI}	Input Leakage Current		±10	μA	$0V < V_{IN} < V_{CC}$
I_{LO}	Output Leakage Current		±10	μA	$0.45 < V_{OUT} < V_{CC}$
V_{CLI}	CLK Input Low Voltage	−0.5	+0.6	volts	
V_{CHI}	CLK Input High Voltage	3.9	V_{CC} +1.0	volts	
C_{IN}	Input Capacitance		10	pF	
C_{IO}	I/O Capacitance		20	pF	

AC CHARACTERISTICS

(V_{CC} = 5V ± 10%, T_A = 0°C to 70°C, T_{CASE} = 0°C to +85°C)

TIMING REQUIREMENTS

Symbol	Parameter	Min	Max	Units	Notes
TCLCL	Clock Period	125	500	ns	
TCLCH	Clock LOW Time	½TCLCL-7.5		ns	
TCHCL	Clock HIGH Time	½TCLCL-7.5		ns	
TARYHCL	ARDY Active Setup Time	20		ns	
TCHARYL	ARDY Hold Time	15		ns	8
TARYLCH	ARDY Inactive Setup Time	35		ns	
TSRYHCL	SRDY Input Setup Time	65,50		ns	1
TSVCH	STATUS Active Setup Time	55		ns	
TSXCL	STATUS Inactive Setup Time	50		ns	
TQIVCL	QS0I, QS1I Setup Time	15		ns	
THAVGV	HLDA Setup Time	50		ns	
TSHVCL	SYSHOLD Asynchronous Setup Time	25		ns	
TGVCH	$\overline{RQ}/\overline{GT}$ Input Setup Time	0		ns	6

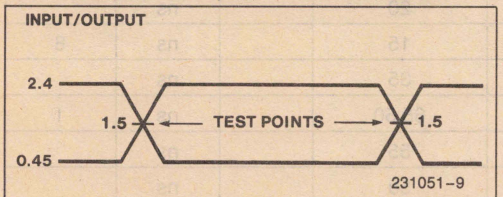
TIMING RESPONSES

Symbol	Parameter	Min	Max	Units	Notes
TSVLH	STATUS Valid to ALE Delay		30	ns	4
TCHLL	ALE Inactive Delay		30	ns	
TCLML	$\overline{RD}$, $\overline{WR}$ Active Delay	10	70	ns	
TCLMH	$\overline{RD}$, $\overline{WR}$ Inactive Delay	10	55	ns	
TSVDTV	STATUS to DT/$\overline{R}$ Delay		30	ns	3
TCLDTV	DT/$\overline{R}$ Active Delay		55	ns	3
TCHDNV	$\overline{DEN}$ Active Delay	10	55	ns	
TCHDNX	$\overline{DEN}$ Inactive Delay	10	55	ns	
TCLQOV	QS0O, QS1O Delay	5	50	ns	
TCHHV	HOLD Delay		50	ns	2,6
TCLSAV	SYSHLDA Delay		50	ns	6
TCLGV	$\overline{RQ}$/$\overline{GT}$ Output Delay		40	ns	6
TGVHV	$\overline{RQ}$/$\overline{GT}$0 To HOLD Delay		50	ns	2,6
TCLLH	ALE Active Delay		30	ns	4
TAELCV	Command Enable Delay		40	ns	
TAEHCX	Command Disable Delay		40	ns	
TCHRO	SRO Output Delay	5	30	ns	5,6
TSRYHRO	SRDY To SRO Delay		30	ns	5
TCSICSO	$\overline{CSIN}$ To $\overline{CSOUT}$ Delay		30	ns	
TCLCSOV	CLK Low to $\overline{CSOUT}$ Delay	10		ns	
TCLCSOH	CLK Low to $\overline{CSOUT}$ Inactive Delay	10		ns	

NOTES (applicable to both spec listing and timing diagrams):

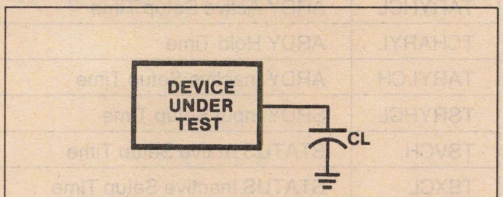
1. TSRYHOL = (80186's) TSRYCL + 30 ns = 65 ns for 6 MHz operation and 50 ns for 8 MHz operation.
2. Timing not tested.
3. DT/$\overline{R}$ will be asserted to the latest of TSVDTV & TCLDTV.
4. ALE will be asserted to the latest of TSVLH & TCLLH.
5. SRO will be asserted to the latest of TCHRO & TSRYHRO.
6. CL = 20–100 pF
7. Address/Data bus shown for reference only.
8. The falling edge of ARDY must be synchronized to CLK.

A.C. TESTING INPUT, OUTPUT WAVEFORM

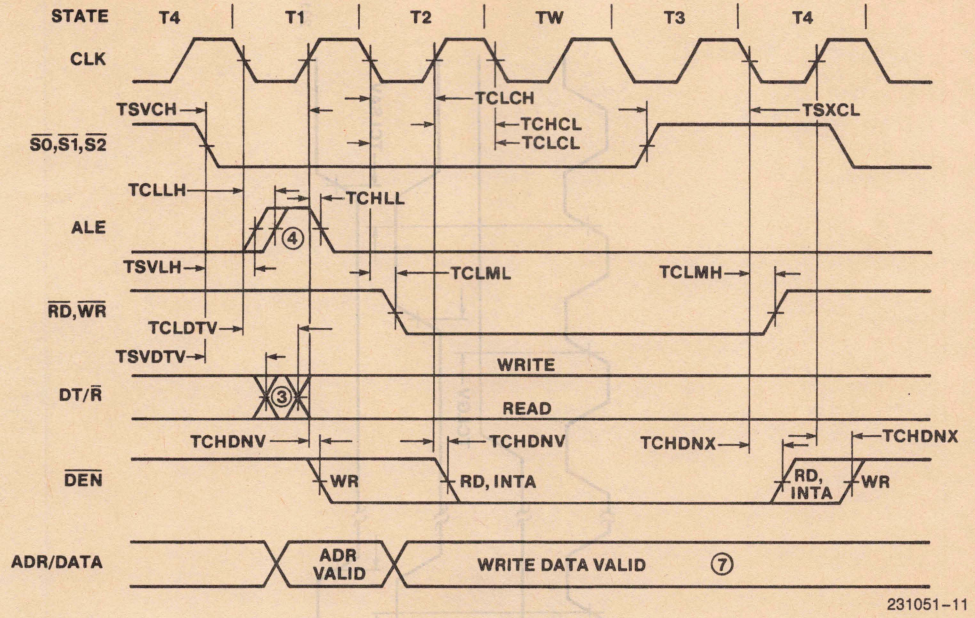


A.C. Testing: Inputs are driven at 2.4V for a Logic '1' and 0.45V for a Logic '0'.

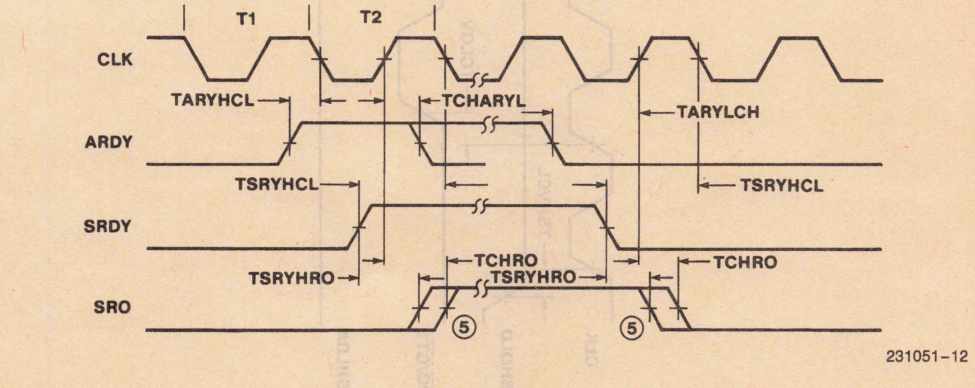
A.C. TESTING LOAD CIRCUIT



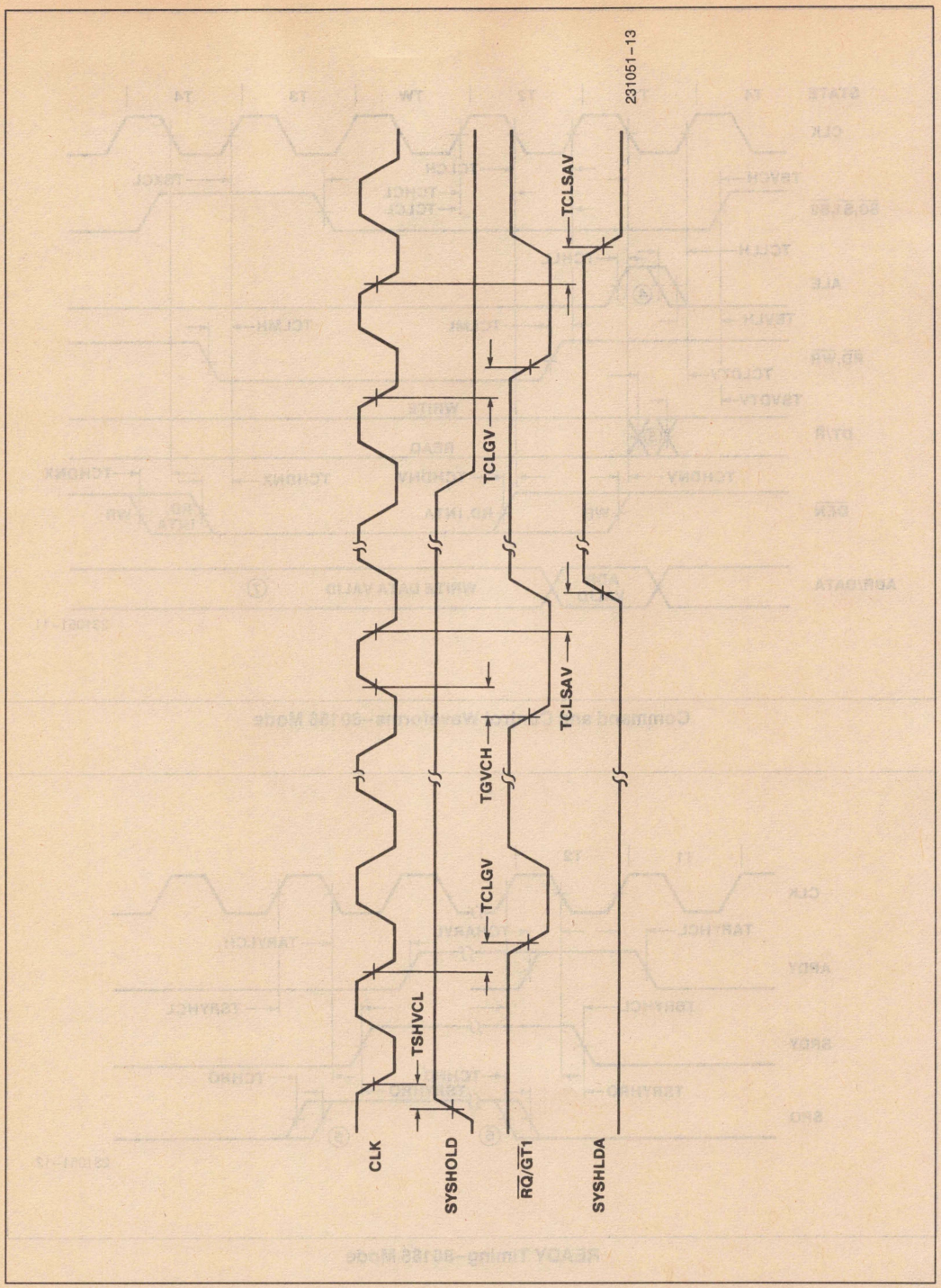
C_L includes Jig Capacitance
C_L = 20–200 pF unless otherwise noted



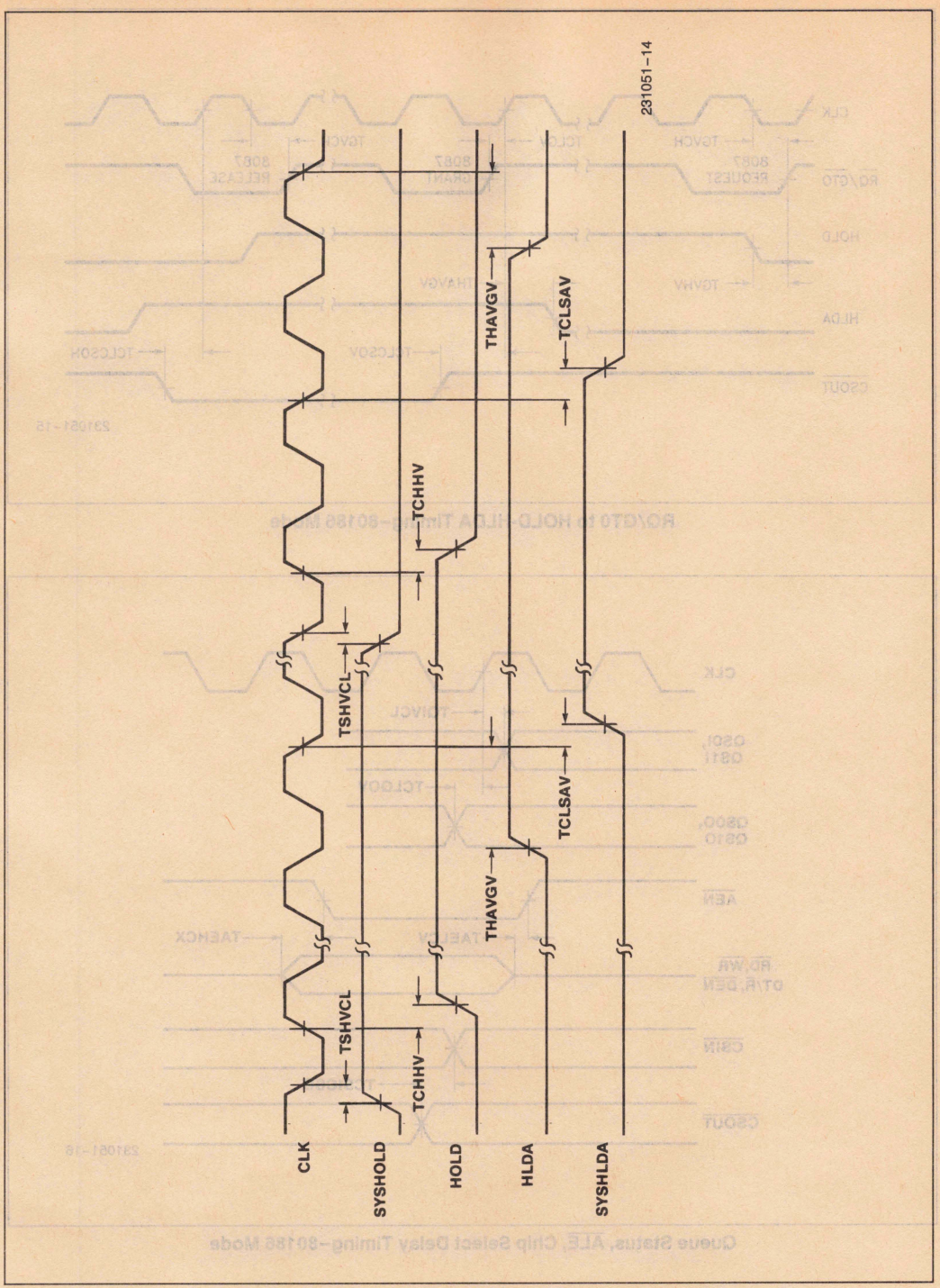
Command and Control Waveforms–80186 Mode



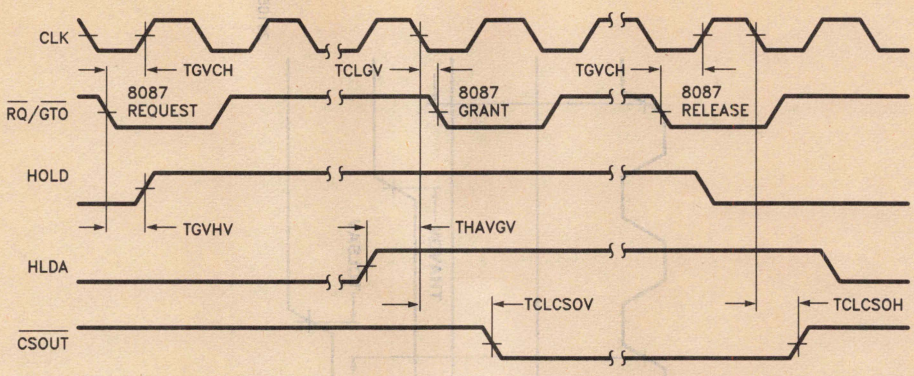
READY Timing–80186 Mode



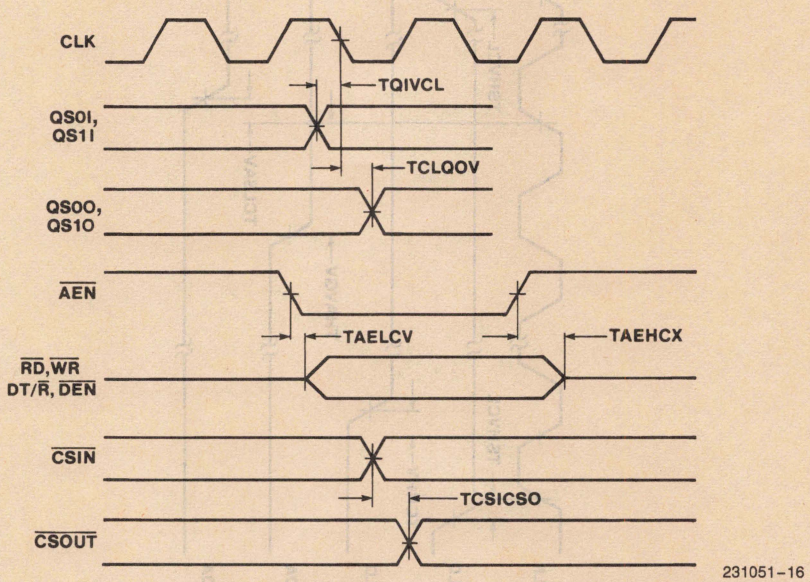
SYSHOLD-SYSHLDA to $\overline{RQ}/\overline{GT1}$ Timing–80186 Mode and 8086 Mode



SYSHOLD-SYSHLDA To HOLD-HLDA Timing–80186 Mode



RQ/GT0 to HOLD-HLDA Timing–80186 Mode



Queue Status, ALE, Chip Select Delay Timing–80186 Mode

80186 Application Notes 18

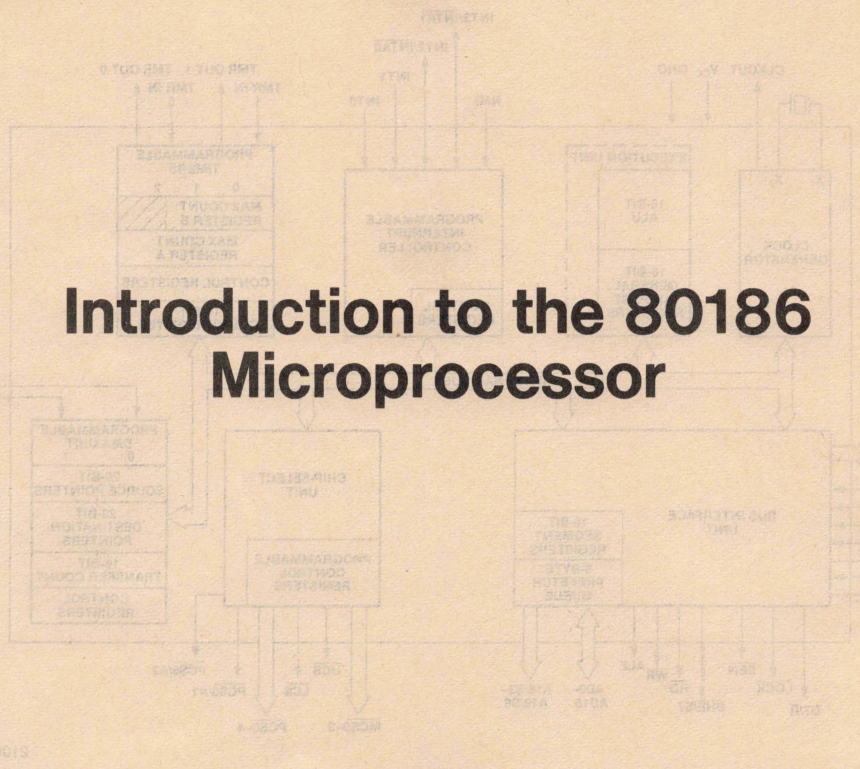
intel®

APPLICATION NOTE

AP-186

September 1986

Introduction to the 80186 Microprocessor



KEN SHOEMAKER
APPLICATIONS ENGINEER

Order Number: 210973-003

1.0 INTRODUCTION

As state of the art technology has increased the number of transistors possible on a single integrated circuit, these devices have attained new, higher levels of both performance and functionality. Riding this crest are the Intel 80186 and 80286 microprocessors. While the 80286 has added memory protection and management to the basic 8086 architecture, the 80186 has integrated six separate functional blocks into a single device.

The purpose of this note is to explain, through example, the use of the 80186 with various peripheral and memory devices. Because the 80186 integrates a DMA unit, timer unit, interrupt controller unit, bus controller unit and chip select and ready generation unit with the CPU on a single chip (see Figure 1), system construction is simplified since many of the peripheral interfaces are integrated onto the device.

The 80186 family actually consists of two processors: the 80186 and 80188. The only difference between the two processors is that the 80186 maintains a 16-bit external data bus while the 80188 has an 8-bit external data bus. Internally, they both implement the same processor with the same integrated peripheral components. Thus, except where noted, all 80186 information in this note also applies to the 80188. The implications of having an 8-bit external data bus on the 80188 are explicitly noted in Appendix I. Any parametric values included in this note are taken from iAPX 186 Advance Information data sheet, and pertain to 8 MHz devices.

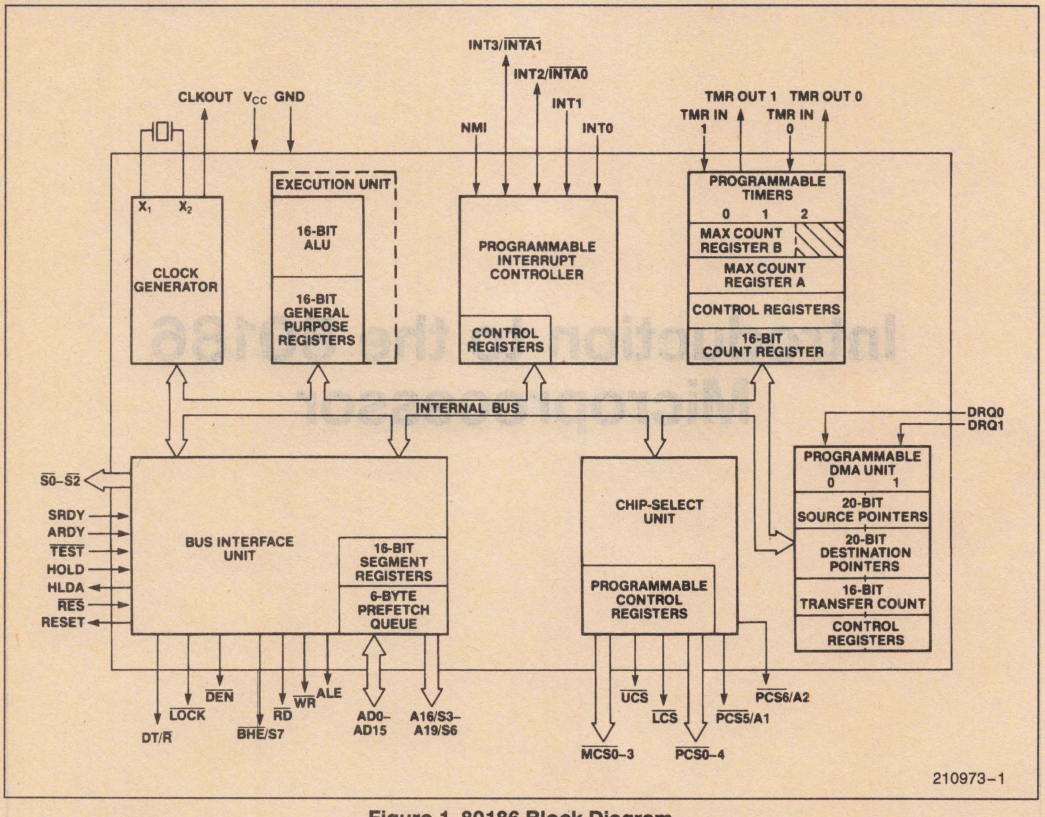


Figure 1. 80186 Block Diagram

2.0 OVERVIEW OF THE 80186

2.1 The CPU

The 80186 CPU shares a common base architecture with the 8086, 8088 and 80286. It is completely object code compatible with the 8086/88. This architecture features four 16-bit general purpose registers (AX, BX, CX, DX) which may be used as operands in most arithmetic operations in either 8 or 16 bit units. It also features four 16-bit "pointer" registers (SI, DI, BP, SP) which may be used both in arithmetic operations and in accessing memory based variables. Four 16-bit segment registers (CS, DS, SS, ES) are provided allowing simple memory partitioning to aid construction of modular programs. Finally, it has a 16-bit instruction pointer and a 16-bit status register.

Physical memory addresses are generated by the 80186 identically to the 8086. The 16-bit segment value is left shifted 4 bits and then is added to an offset value which is derived from combinations of the pointer registers, the instruction pointer, and immediate values (see Figure 2). Any carry out of this addition is ignored. The result of this addition is a 20-bit physical address which is presented to the system memory.

The 80186 has a 16-bit ALU which performs 8 or 16-bit arithmetic and logical operations. It provides for data movement among registers, memory and I/O space. In addition, the CPU allows for high speed data transfer from one area of memory to another using string move instructions, and to or from an I/O port and memory using block I/O instructions. Finally, the CPU provides a wealth of conditional branch and other control instructions.

In the 80186, as in the 8086, instruction fetching and instruction execution are performed by separate units: the bus interface unit and the execution unit, respectively. The 80186 also has a 6-byte prefetch queue as does the 8086. The 80188 has a 4-byte prefetch queue as does the 8088. As a program is excecuting, opcodes are fetched from memory by the bus interface unit and placed in this queue. Whenever the execution unit requires another instruction, it takes it out of the queue. Effective processor throughput is increased by adding this queue, since the bus interface unit may continue to fetch instructions while the execution unit executes a long instruction. Then, when the CPU completes this instruction, it does not have to wait for another instruction to be fetched from memory.

2.2 80186 CPU Enhancements

Although the 80186 is completely object code compatible with the 8086, most of the 8086 instructions require fewer clock cycles to execute on the 80186 than on the 8086 because of hardware enhancements in the bus interface unit and the execution unit. In addition, the 80186 provides many new instructions which simplify assembly language programming, enhance the performance of high level language implementations, and reduce object code sizes for the 80186. These new instructions are also included in the 80286. A complete description of the architecture and instruction execution of the 80186 can be found in volume I of the iAPX86/186 users manual. The algorithm for the new instructions are also given in appendix H of this note.

2.3 DMA Unit

The 80186 includes a DMA unit which provides two high speed DMA channels. This DMA unit will per-

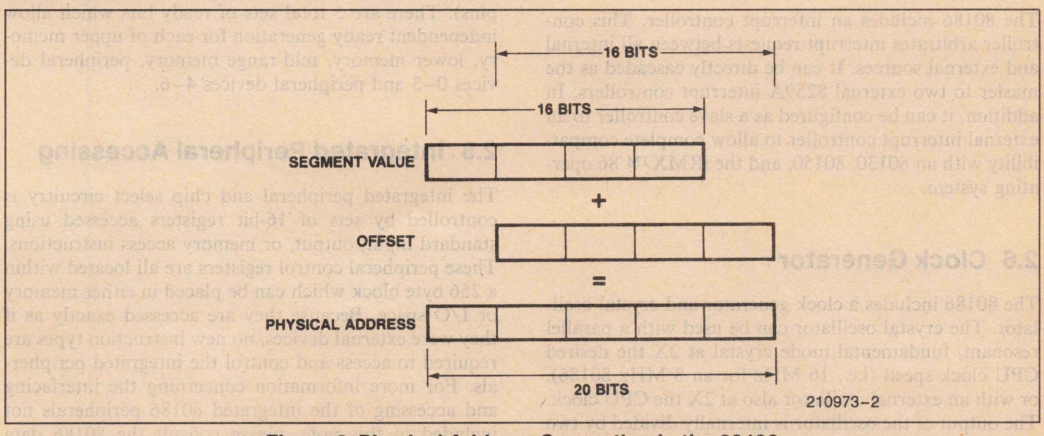


Figure 2. Physical Address Generation in the 80186

form transfers to or from any combination of I/O space and memory space in either byte or word units. Every DMA cycle requires two to four bus cycles, one or two to fetch the data to an internal register, and one or two to deposit the data. This allows word data to be located on odd boundaries, or byte data to be moved from odd locations to even locations. This is normally difficult, since odd data bytes are transferred on the upper 8 data bits of the 16-bit data bus, while even data bytes are transferred on the lower 8 data bits of the data bus.

Each DMA channel maintains independent 20-bit source and destination pointers which are used to access the source and destination of the data transferred. Each of these pointers may independently address either I/O or memory space. After each DMA cycle, the pointers may be independently incremented, decremented, or maintained constant. Each DMA channel also maintains a transfer count which may be used to terminate a series of DMA transfers after a pre-programmed number of transfers.

2.4 Timers

The 80186 includes a timer unit which contains 3 independent 16-bit timer/counters. Two of these timers can be used to count external events, to provide waveforms derived from either the CPU clock or an external clock of any duty cycle, or to interrupt the CPU after a specified number of timer "events". The third timer counts only CPU clocks and can be used to interrupt the CPU after a programmable number of CPU clocks, to give a count pulse to either or both of the other two timers after a programmable number of CPU clocks, or to give a DMA request pulse to the integrated DMA unit after a programmable number of CPU clocks.

2.5 Interrupt Controller

The 80186 includes an interrupt controller. This controller arbitrates interrupt requests between all internal and external sources. It can be directly cascaded as the master to two external 8259A interrupt controllers. In addition, it can be configured as a slave controller to an external interrupt controller to allow complete compatibility with an 80130, 80150, and the iRMX™ 86 operating system.

2.6 Clock Generator

The 80186 includes a clock generator and crystal oscillator. The crystal oscillator can be used with a parallel resonant, fundamental mode crystal at 2X the desired CPU clock speed (i.e., 16 MHz for an 8 MHz 80186), or with an external oscillator also at 2X the CPU clock. The output of the oscillator is internally divided by two to provide the 50% duty cycle CPU clock from which all 80186 system timing derives. The CPU clock is externally available, and all timing parameters are referenced to this externally available signal. The clock generator also provides ready synchronization for the processor.

2.7 Chip Select and Ready Generation Unit

The 80186 includes integrated chip select logic which can be used to enable memory or peripheral devices. Six output lines are used for memory addressing and seven output lines are used for peripheral addressing.

The memory chip select lines are split into 3 groups for separately addressing the major memory areas in a typical 8086 system: upper memory for reset ROM, lower memory for interrupt vectors, and mid-range memory for program memory. The size of each of these regions is user programmable. The starting location and ending location of lower memory and upper memory are fixed at 00000H and FFFFFH respectively; the starting location of the mid-range memory is user programmable.

Each of the seven peripheral select lines address one of seven contiguous 128 byte blocks above a programmable base address. This base address can be located in either memory or I/O space in order that peripheral devices may be I/O or memory mapped.

Each of the programmed chip select areas has associated with it a set of programmable ready bits. These ready bits control an integrated wait state generator. This allows a programmable number of wait states (0 to 3) to be automatically inserted whenever an access is made to the area of memory associated with the chip select area. In addition, each set of ready bits includes a bit which determines whether the external ready signals (ARDY and SRDY) will be used, or whether they will be ignored (i.e., the bus cycle will terminate even though a ready has not been returned on the external pins). There are 5 total sets of ready bits which allow independent ready generation for each of upper memory, lower memory, mid-range memory, peripheral devices 0–3 and peripheral devices 4–6.

2.8 Integrated Peripheral Accessing

The integrated peripheral and chip select circuitry is controlled by sets of 16-bit registers accessed using standard input, output, or memory access instructions. These peripheral control registers are all located within a 256 byte block which can be placed in either memory or I/O space. Because they are accessed exactly as if they were external devices, no new instruction types are required to access and control the integrated peripherals. For more information concerning the interfacing and accessing of the integrated 80186 peripherals not included in this note, please consult the 80186 data sheet, or the iAPX 86/186 User's Manual Hardware Reference.

3.0 USING THE 80186

3.1 Bus Interfacing to the 80186

3.1.1 OVERVIEW

The 80186 bus structure is very similar to the 8086 bus structure. It includes a multiplexed address/data bus, along with various control and status lines (see Table 1). Each bus cycle requires a minimum of 4 CPU clock cycles along with any number of wait states required to accommodate the speed access limitations of external memory or peripheral devices. The bus cycles initiated by the 80186 CPU are identical to the bus cycles intitiated by the 80186 integrated DMA unit.

In the following discussion, all timing values given are for an 8 MHz 80186. Future speed selections of the part may have different values for the various parameters.

Each clock cycle of the 80186 bus cycle is called a "T" state, and are numbered sequentially T_1, T_2, T_3, T_W and T_4. Additional idle T states (T_i) can occur between T_4 and T_1 when the processor requires no bus activity (instruction fetches, memory writes, I/O reads, etc.). The ready signals control the number of wait states (t_W) inserted in each bus cycle. This number can vary from 0 to positive infinity.

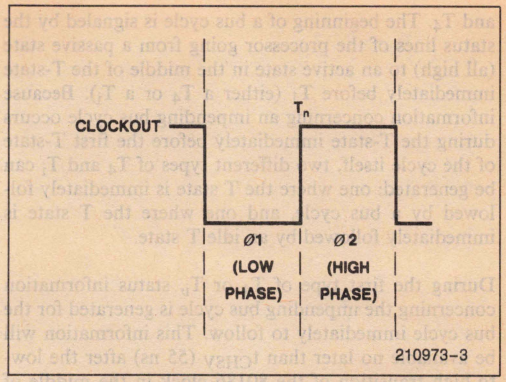


Figure 3. T-state in the 80186

The beginning of a T state is signaled by a high to low transition of the CPU clock. Each T state is divided into two phases, phase 1 (or the low phase) and phase 2 (or the high phase) which occur during the low and high levels of the CPU clock respectively (see Figure 3).

Different types of bus activity occur for all of the T-states (see Figure 4). Address generation information occurs during T_1, data generation during T_2, T_3, T_W

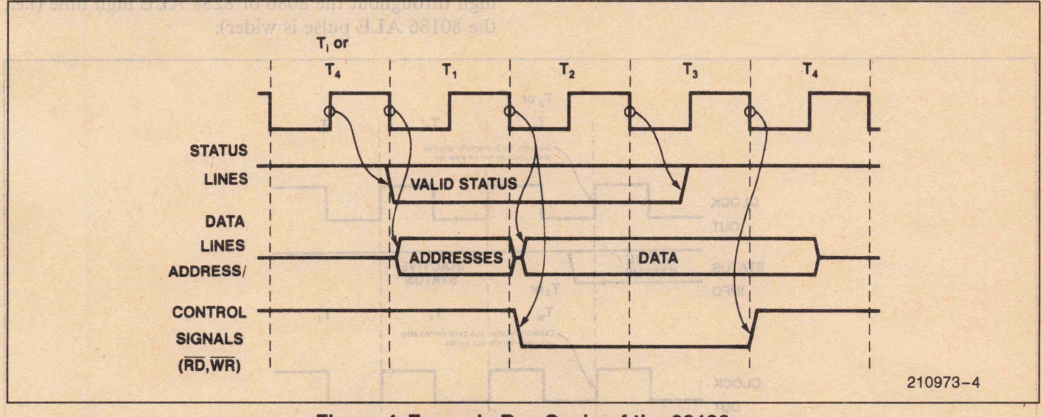


Figure 4. Example Bus Cycle of the 80186

Table 1. 80186 Bus Signals

Function	Signal Name
address/data	AD0–AD15
address/status	A16/S3-A19-S6, $\overline{BHE}$/S7
co-processor control	$\overline{TEST}$
local bus arbitration	HOLD, HLDA
local bus control	ALE, $\overline{RD}$, $\overline{WR}$, DT/$\overline{R}$, $\overline{DEN}$
multi-master bus	$\overline{LOCK}$
ready (wait) interface	SRDY, ARDY
status information	$\overline{S0}$–$\overline{S2}$

and T_4. The beginning of a bus cycle is signaled by the status lines of the processor going from a passive state (all high) to an active state in the middle of the T-state immediately before T_1 (either a T_4 or a T_i). Because information concerning an impending bus cycle occurs during the T-state immediately before the first T-state of the cycle itself, two different types of T_4 and T_i can be generated: one where the T state is immediately followed by a bus cycle, and one where the T state is immediately followed by an idle T state.

During the first type of T_4 or T_i, status information concerning the impending bus cycle is generated for the bus cycle immediately to follow. This information will be available no later than t_{CHSV} (55 ns) after the low-to-high transition of the 80186 clock in the middle of the T state. During the second type of T_4 or T_i the status outputs remain inactive (high), since no bus cycle is to be started. This means that the decision per the nature of a T_4 or T_i state (i.e., whether it is immediately followed by a T_i or a T_1) is decided at the beginning of the T-state immediately preceding the T_4 or T_i (see Figure 5). This has consequences for the bus latency time (see section 3.3.2 on bus latency).

3.1.2. PHYSICAL ADDRESS GENERATION

Physical addresses are generated by the 81086 during T_1 of a bus cycle. Since the address and data lines are multiplexed on the same set of pins, addresses must be latched during T_1 if they are required to remain stable for the duration of the bus cycle. To facilitate latching of the physical address, the 80186 generates an active high ALE (Address Latch Enable) signal which can be directly connected to a transparent latch's strobe input.

Figure 6 illustrates the physical address generation parameters of the 80186. Addresses are guaranteed valid no greater than t_{CLAV} (44 ns) after the beginning of T_1, and remain valid at least t_{CLAX} (10 ns) after the end of T_1. The ALE signal is driven high in the middle of the T state (either T_4 or T_i) immediately preceding T_1 and is driven low in the middle of T_1, no sooner than t_{AVAL} (30 ns) after addresses become valid. This parameter (t_{AVAL}) is required to satisfy the address latch set-up times of address valid until strobe inactive. Addresses remain stable on the address/data bus at least t_{LLAX} (30 ns) after ALE goes inactive to satisfy address latch hold times of strobe inactive to address invalid.

Because ALE goes high long before addresses become valid, the delay through the address latches will be chiefly the propagation delay through the latch rather than the delay from the latch strobe, which is typically longer than the propagation delay. For the Intel 8282 latch, this parameter is t_{IVOV}, the input valid to output valid delay when strobe is held active (high). Note that the 80186 drives ALE high one full clock phase earlier than the 8086 or the 8288 bus controller, and keeps it high throughout the 8086 or 8288 ALE high time (i.e., the 80186 ALE pulse is wider).

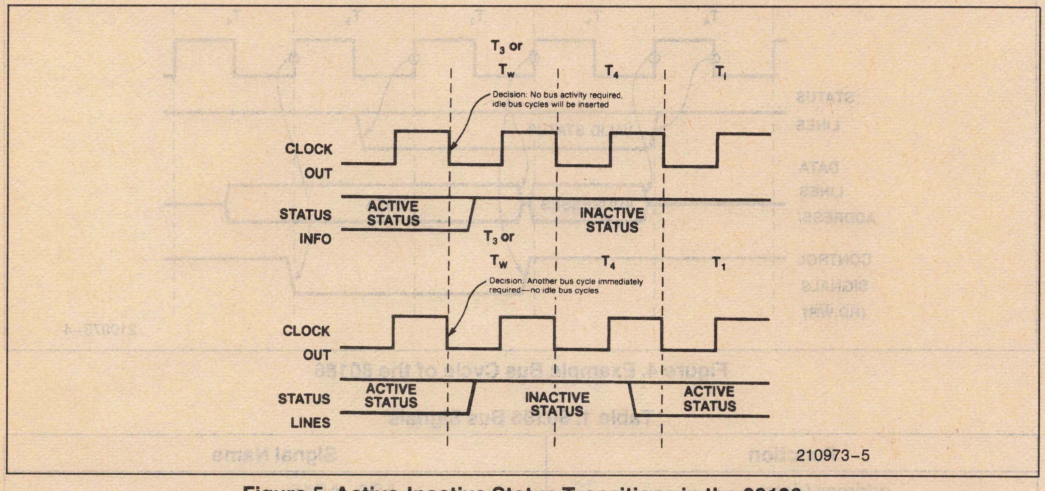
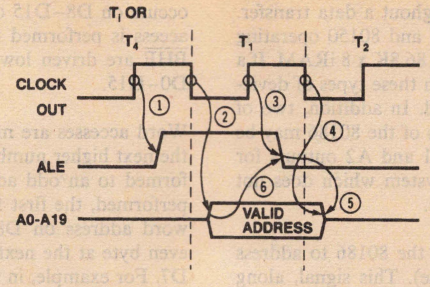


Figure 5. Active-Inactive Status Transitions in the 80186



NOTES:
1. t_{CHLH}: Clock high to ALE high = 35 ns max
2. t_{CLAV}: Clock low to address valid = 44 ns max
3. t_{CHLL}: Clock high to ALE low = 35 ns max
4. t_{CLAX}: Clock low to address invalid (address hold from clock low) = 10 ns min
5. t_{LLAX}: ALE low to address invalid (address hold from ALE) = 30 ns min
6. t_{AVAL}: Address valid to ALE low (address setup to ALE) = 30 ns min

Figure 6. Address Generation Timing of the 80186

A typical circuit for latching physical addresses is shown in Figure 7. This circuit uses 3 8282 transparent octal non-inverting latches to demultiplex all 20 address bits provided by the 81086. Typically, the upper 4 address bits are used only to select among various memory components or subsystems, so when the integrated chip selects (see section 8) are used, these upper bits need not be latched. The worst case address generation time from the beginning of T_1 (including address latch propagation time (t_{IVOV}) of the Intel 8282) for the circuit is:

$$t_{CLAV}\ (44\ ns) + t_{IVOV}\ (30\ ns) = 74\ ns$$

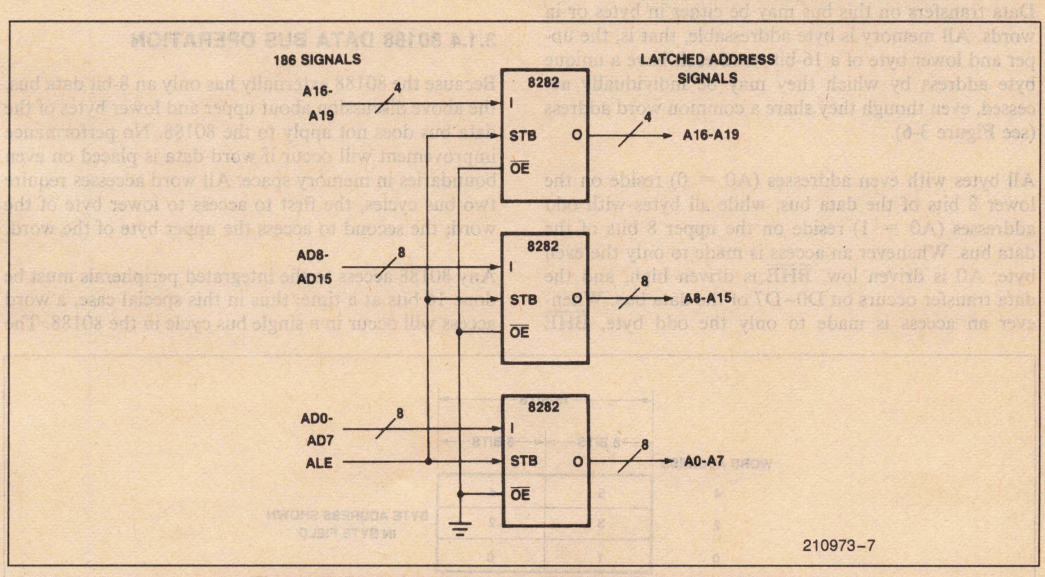


Figure 7. Demultiplexing the Address Bus of the 80186

Many memory or peripheral devices may not require addresses to remain stable throughout a data transfer. Examples of these are the 80130 and 80150 operating system firmware chips, and the 2186 8K x 8 iRAM. If a system is constructed wholly with these types of devices, addresses need not be latched. In addition, two of the peripheral chip select outputs of the 80186 may be configured to provide latched A1 and A2 outputs for peripheral register selects in a system which does not demultiplex the address/data bus.

One more signal is generated by the 80186 to address memory: $\overline{BHE}$ (Bus High Enable). This signal, along with A0, is used to enable byte devices connected to either or both halves (bytes) of the 16-bit data bus (see section 3.1.3 on data bus operation section). Because A0 is used only to enable devices onto the lower half of the data bus, memory chip address inputs are usually driven by address bits A1–A19, NOT A0–A19. This provides 512K unique *word* addresses, or 1M unique BYTE addresses.

Of course, $\overline{BHE}$ is not present on the 8 bit 80188. All data transfers occur on the 8 bits of the data bus.

3.1.3 80186 DATA BUS OPERATION

Throughout T_2, T_3, T_W and T_4 of a bus cycle the multiplexed address/data bus becomes a 16-bit data bus. Data transfers on this bus may be either in bytes or in words. All memory is byte addressable, that is, the upper and lower byte of a 16-bit word each have a unique byte address by which they may be individually accessed, even though they share a common word address (see Figure 3-6).

All bytes with even addresses (A0 = 0) reside on the lower 8 bits of the data bus, while all bytes with odd addresses (A0 = 1) reside on the upper 8 bits of the data bus. Whenever an access is made to only the even byte, A0 is driven low, $\overline{BHE}$ is driven high, and the data transfer occurs on D0–D7 of the data bus. Whenever an access is made to only the odd byte, $\overline{BHE}$ is driven low, A0 is driven high, and the data transfer occurs on D8–D15 of the data bus. Finally, if a word access is performed to an even address, both A0 and $\overline{BHE}$ are driven low and the data transfer occurs on D0–D15.

Word accesses are made to the addressed byte and to the next higher numbered byte. If a word access is performed to an odd address, two byte accesses must be performed, the first to access the odd byte at the first word address on D8–D15, the second to access the even byte at the next sequential word address on D0–D7. For example, in Figure 8, byte 0 and byte 1 can be individually accessed (read or written) in two separate bus cycles (byte accesses) to byte addresses 0 and 1 at word address 0. They may also be accessed together in a single bus cycle (word access) to word address 0. However, if a word access is made to address 1, two bus cycles will be required, the first to access byte 1 at word address 0 (note byte 0 will not be accessed), and the second to access byte 2 at word address 2 (note byte 3 will not be accessed). This is why all word data should be located at even addresses to maximize processor performance.

When byte reads are made, the data returned on the half of the data bus not being accessed is ignored. When byte writes are made, the data driven on the half of the data bus not being written is indeterminate.

3.1.4 80188 DATA BUS OPERATION

Because the 80188 externally has only an 8-bit data bus, the above discussion about upper and lower bytes of the data bus does not apply to the 80188. No performance improvement will occur if word data is placed on even boundaries in memory space. All word accesses require two bus cycles, the first to access to lower byte of the word; the second to access the upper byte of the word.

Any 80188 access to the integrated peripherals must be done 16 bits at a time: thus in this special case, a word access will occur in a single bus cycle in the 80188. The

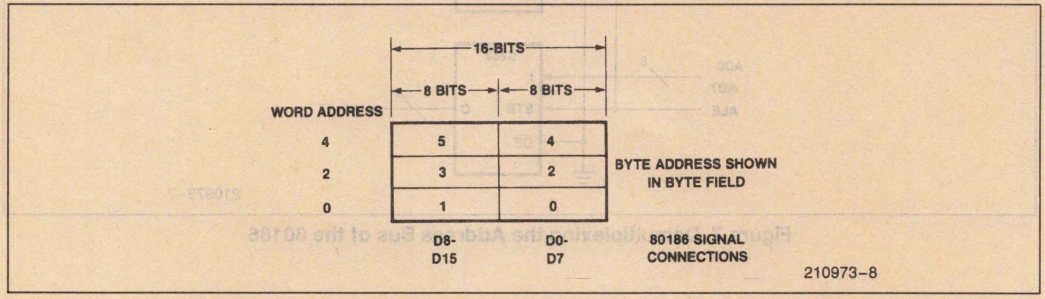


Figure 8. Physical Memory Byte/Word Addressing in the 80186

external data bus will record only a single byte being transferred, however.

3.1.5 GENERAL DATA BUS OPERATION

Because of the bus drive capabilities of the 80186 (200 pF, sinking 2 mA, sourcing 400 μA, roughly twice that of the 8086), this bus may not require additional buffering in many small systems. If data buffers are not used in the system, care should be taken not to allow bus contention between the 80186 and the devices directly connected to the 80186 data bus. Since the 80186 floats the address/data bus before activating any command lines, the only requirement on a directly connected device is that it floats its output drivers after a read *BEFORE* the 80186 beings to drive address information for the next bus cycle. The parameter of interest here is the minimum time from $\overline{RD}$ inactive until addresses active for the next bus cycle (t_{RHAV}) which has a minimum value of 85 ns. If the memory or peripheral device cannot disable its output drivers in this time, data buffers will be required to prevent both the 80186 and the peripheral or memory device from driving these lines concurrently. Note, this parameter is unaffected by the addition of wait states. Data buffers solve this problem because their output float times are typically much faster than the 80186 required minimum.

If the buffers are required, the 80186 provides a $\overline{DEN}$ (Data ENable) and DT/$\overline{R}$ (Data Transmit/Receive) signals to simplify buffer interfacing. The $\overline{DEN}$ and DT/$\overline{R}$ signals are activated during all bus cycles, whether or not the cycle addresses buffered devices. The $\overline{DEN}$ signal is driven low whenever the processor is either ready to receive data (during a read) or when the processor is ready to send data (during a write) (that is, any time during an active bus cycle when address information is not being generated on the address/data pins). In most systems, the $\overline{DEN}$ signal should NOT be directly connected to the $\overline{OE}$ input of buffers, since unbuffered devices (or other buffers) may be directly connected to the processor's address/data pins. If $\overline{DEN}$ were directly connected to several buffers, contention would occur during read cycles, as many devices attempt to drive the processor bus. Rather, it should be a factor (along with the chip selects for buffered devices) in generating the output enable input of a bi-directional buffer.

The DT/$\overline{R}$ signal determines the direction of data propagation through the bi-directional bus buffers. It is high whenever data is being driven out from the processor, and is low whenever data is being read into the processor. Unlike the $\overline{DEN}$ signal, it may be directly connected to bus buffers, since this signal does not usually directly enable the output drivers of the buffer. An example data bus subsystem supporting both buffered and unbuffered devices is shown in Figure 9. Note that the A side of the 8286 buffer is connected to the 80186, the B side to the external device. The B side of the buffer has greater drive capacity than the A side (since it is meant to drive much greater loads). The DT/$\overline{R}$ signal can directly drive the T (transmit) signal of the buffer, since it has the correct polarity for this configuration.

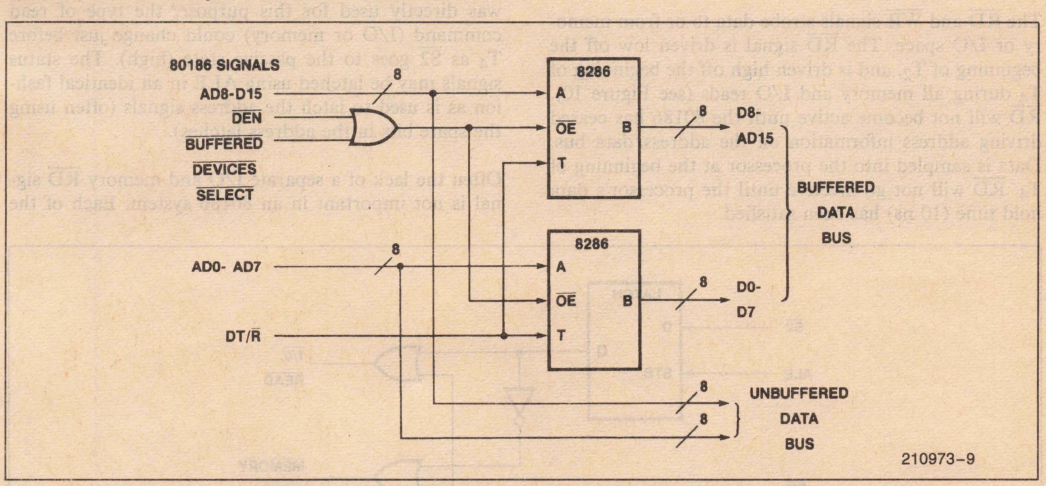
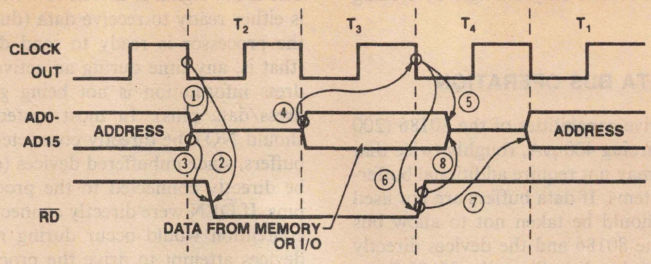


Figure 9. Example 80186 Buffered/Unbuffered Data Bus



NOTES:
1. t_{CLAZ}: Clock low until address float = 35 ns max
2. t_{CLRL}: Clock low until $\overline{RD}$ active = 70 ns max
3. t_{AZRL}: Address float until $\overline{RD}$ active = 0 ns min
4. t_{DVCL}: Data valid until clock low (data input set-up time) = 20 ns min*
5. t_{CLDX}: Clock low unitl data invalid (data input hold time from clock) = 10 ns min*
6. t_{CLRH}: Clock low until $\overline{RD}$ high = 10 ns min
7. t_{RHAV}: $\overline{RD}$ high until addresses valid = 85 ns min
8. t_{RHDX}: Read high until data invalid (data input hold from $\overline{RD}$) = 0 ns min*

*Input requirements of 80186, all others are output characteristics

Figure 10. Read Cycle Timing of the 80186

3.1.6 CONTROL SIGNALS

The 80186 directly provides the control signals $\overline{RD}$, $\overline{WR}$, $\overline{LOCK}$ and $\overline{TEST}$. In addition, the 80186 provides the status signals S0–S2 and S6 from which all other required bus control signals can be generated.

3.1.6.1 $\overline{RD}$ and $\overline{WR}$

The $\overline{RD}$ and $\overline{WR}$ signals strobe data to or from memory or I/O space. The $\overline{RD}$ signal is driven low off the beginning of T_2, and is driven high off the beginning of T_4 during all memory and I/O reads (see Figure 10). $\overline{RD}$ will not become active until the 80186 has ceased driving address information on the address/data bus. Data is sampled into the processor at the beginning of T_4. $\overline{RD}$ will not go inactive until the processor's data hold time (10 ns) has been satisfied.

Note that the 80186 does not provide separate I/O and memory $\overline{RD}$ signals. If separate I/O read and memory read signals are required, they can be synthesized using the $\overline{S2}$ signal (which is low for all I/O operations and high for all memory operations) and the $\overline{RD}$ signal (see Figure 11). It should be noted that if this approach is used, the $\overline{S2}$ signal will require latching, since the $\overline{S2}$ signal (like $\overline{S0}$ and $\overline{S1}$) goes to a passive state well before the beginning of T_4 (where $\overline{RD}$ goes inactive). If $\overline{S2}$ was directly used for this purpose, the type of read command (I/O or memory) could change just before T_4 as $\overline{S2}$ goes to the passive state (high). The status signals may be latched using ALE in an identical fashion as is used to latch the address signals (often using the spare bits in the address latches).

Often the lack of a separate I/O and memory $\overline{RD}$ signal is not important in an 80186 system. Each of the

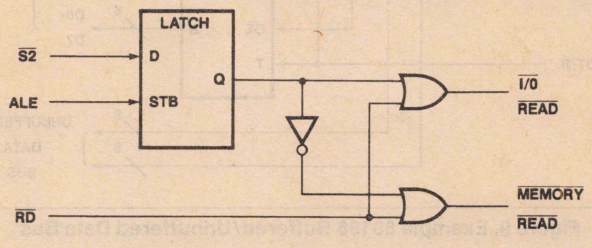
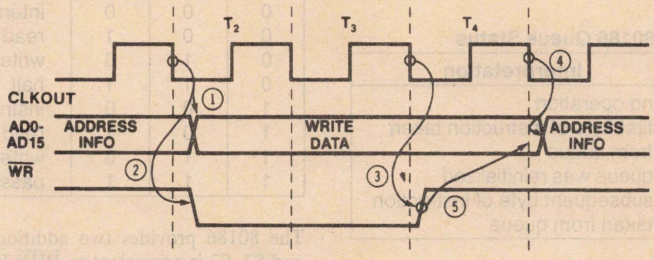


Figure 11. Generating I/O and Memory Read Signals from the 80186

81086 chip select signals will respond on only one of memory or I/O accesses (the memory chip selects respond only to accesses memory space; the peripheral chip selects can respond to accesses in either I/O or memory space, at programmer option). Thus, the chip select signal enables the external device only during accesses to the proper address in the proper space.

The $\overline{WR}$ signal is also driven low off the beginning of T_2 and driven high off the beginning of T_4. Like the $\overline{RD}$ signal, the $\overline{WR}$ signal is active for all memory and I/O writes, and also like the $\overline{RD}$ signal, separate I/O and memory writes may be generated using the latched $\overline{S2}$ signal along with the $\overline{WR}$ signal (see Figure 12).

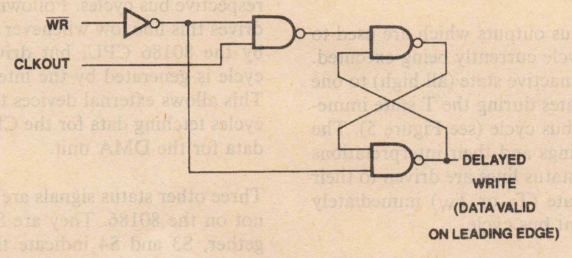
More importantly, however, is the active going edge of write. At the time $\overline{WR}$ makes its active (high to low) transition, valid write data is NOT present on the data bus. This has consequences when using this signal as a write enable signal for DRAMs and iRAMs since both of these devices require that the write data be stable on the data bus at the time of the inactive to active transition of the $\overline{WE}$ signal. In DRAM applications, this problem is solved by a DRAM controller (such as the Intel 8207 or 8203), while with iRAMs this problem may be solved by placing cross-coupled NAND gates between the CPU and the iRAMS on the $\overline{WR}$ line (see Figure 13). This will delay the active going edge of the $\overline{WR}$ signal to the iRAMs by a clock phase, allowing valid data to be driven onto the data bus.



210973-12

NOTES:
1. t_{CLDV}: Clock low until data valid = 44 ns max
2. t_{CVCTV}: Clock low until $\overline{WR}$ active = 70 ns max
3. t_{CVCTX}: Clock low until $\overline{WR}$ inactive = 55 ns max
4. t_{CLDOX}: Clock high until data invalid = 10 ns max
5. $\overline{WR}$ inactive until data invalid = t_{CLCL} min − t_{CVCTX} + t_{CLDOX}
   = 125 − 55 + 10
   = 80 ns

Figure 12. Write Cycle Timing of the 80186



210973-13

Figure 13. Synthesizing Delayed Write from the 80186

3.1.6.2 Queue Status Signals

If the $\overline{RD}$ line is externally grounded during reset and remains grounded during processor operation, the 80186 will enter "queue status" mode. When in this mode, the $\overline{WR}$ and ALE signals become queue status outputs, reflecting the status of the internal prefetch queue during each clock cycle. These signals are provided to allow a processor extension (such as the Intel 8087 floating point processor) to track execution of instructions within the 80186. The interpretation of QS0 (ALE) and QS1 ($\overline{WR}$) are given in Table 2. These signals change on the high-to-low clock transition, one clock phase earlier than on the 8086. Note that since execution unit operation is independent of bus interface unit operation, queue status lines may change in any T state.

Table 2. 80186 Queue Status

QS1	QS0	Interpretation
0	0	no operation
0	1	first byte of instruction taken from queue
1	0	queue was reinitialized
1	1	subsequent byte of instruction taken from queue

Since the ALE, $\overline{RD}$, and $\overline{WR}$ signals are not directly available from the 80186 when it is configured in queue status mode, these signals must be derived from the status lines $\overline{S0}$–$\overline{S2}$ using an external 8288 bus controller (see below). To prevent the 80186 from accidentally entering queue status mode during reset, the $\overline{RD}$ line is internally provided with a weak pullup device. $\overline{RD}$ is the ONLY three-state or input pin on the 80186 which is supplied with a pullup or pulldown device.

3.1.6.3 Status Lines

The 80186 provides 3 status outputs which are used to indicate the type of bus cycle currently being executed. These signals go from an inactive state (all high) to one of seven possible active states during the T state immediately preceding T_1 of a bus cycle (see Figure 5). The possible status line encodings and their interpretations are given in Table 3. The status lines are driven to their inactive state in the T state (T_3 or T_W) immediately preceding T_4 of the current bus cycle.

The status lines may be directly connected to an 8288 bus controller, which can be used to provide local bus control signals or multi-bus control signals (see Figure 14). Use of the 8288 bus controller does not preclude the use of the 80186 generated $\overline{RD}$, $\overline{WR}$ and ALE signals, however. The 80186 directly generated signals, may be used to provide local bus control signals, while an 8288 is used to provide multi-bus control signals, for example.

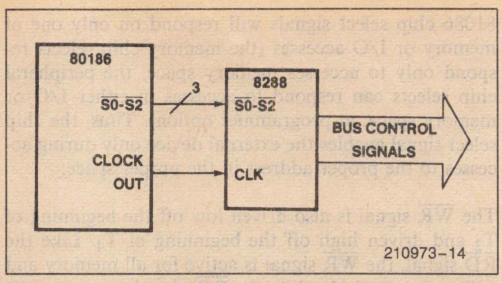


Figure 14. 80186/8288 Bus Controller Interconnection

Table 3. 80186 Status Line Interpretation

$\overline{S2}$	$\overline{S1}$	$\overline{S0}$	Operation
0	0	0	interrupt acknowledge
0	0	1	read I/O
0	1	0	write I/O
0	1	1	halt
1	0	0	instruction fetch
1	0	1	read memory
1	1	0	write memory
1	1	1	passive

The 80186 provides two additional status signals: S6 and S7. S7 is equivalent to $\overline{BHE}$ (see section 3.1.2) and appears on the same pin as $\overline{BHE}$. $\overline{BHE}$/S7 changes state, reflecting the bus cycle about to be run, in the middle of the T state (T_4 or T_i) immediately preceding T_1 of the bus cycle. This means that $\overline{BHE}$/S7 does not need to be latched, i.e., it may be used directly as the $\overline{BHE}$ signal. S6 provides information concerning the unit generating the bus cycle. It is time multiplexed with A19, and is available during T_2, T_3, T_4 and T_W. In the 8086 family, all central processors (e.g., the 8086, 8088 and 8087) drive this line low, while all I/O processors (e.g., 8089) drive this line high during their respective bus cycles. Following this scheme, the 80186 drives this line low whenever the bus cycle is generated by the 80186 CPU, but drives it high when the bus cycle is generated by the integrated 80186 DMA unit. This allows external devices to distinguish between bus cycles fetching data for the CPU from those transfering data for the DMA unit.

Three other status signals are available on the 8086 but not on the 80186. They are S3, S4, and S5. Taken together, S3 and S4 indicate the segment register from which the current physical address drives. S5 indicates the state of the interrupt flip-flop. On the 80186, these signals will ALWAYS be low.

3.1.6.4 $\overline{TEST}$ and $\overline{LOCK}$

Finally, the 80186 provides a $\overline{TEST}$ input and a $\overline{LOCK}$ output. The $\overline{TEST}$ input is used in conjunction with the processor WAIT instruction. It is typically driven by a processor extension (like the 8087) to indicate whether

it is busy. Then, by executing the WAIT (or FWAIT) instruction, the central processor may be forced to temporarily suspend program execution until the processor extension indicates that it is idle by driving the $\overline{TEST}$ line low.

The $\overline{LOCK}$ output is driven low whenever the data cycles of a LOCKED instruction are executed. A LOCKED instruction is generated whenever the LOCK prefix occurs immediately before an instruction. The LOCK prefix is active for the single instruction immediately following the LOCK prefix. This signal is used to indicate to a bus arbiter (e.g., the 8289) that a series of locked data transfers is occurring. The bus arbiter should under no circumstances release the bus while locked transfers are occurring. The 80186 will not recognize a bus HOLD, nor will it allow DMA cycles to be run by the integrated DMA controller during locked data transfers. LOCKED transfers are used in multiprocessor systems to access memory based semaphore variables which control access to shared system resources (see AP-106, "Multiprogramming with the iAPX88 and iAPX86 Microsystems," by George Alexy (Sept. 1980)).

On the 80186, the $\overline{LOCK}$ signal will go active during T_1 of the first DATA cycle of the locked transfer. It is driven inactive 3 T-states after the beginning of the last DATA cycle of the locked transfers. On the 8086, the $\overline{LOCK}$ signal is activated immediately after the LOCK prefix is executed. The LOCK prefix may be executed well before the processor is prepared to perform the locked data transfer. This has the unfortunate consequence of activating the $\overline{LOCK}$ signal before the first LOCKED data cycle is performed. Since $\overline{LOCK}$ is active before the processor requires the bus for the data transfer, opcode pre-fetching can be LOCKED. However, since the 80186 does not activate the $\overline{LOCK}$ signal until the processor is ready to actually perform the locked transfer, locked pre-fetching will not occur with the 80186.

Note that the $\overline{LOCK}$ signal does not remain active until the end of the last data cycle of the locked transfer. This may cause problems in some systems if, for example, the processor requests memory access from a dual ported RAM array and is denied immediate access (because of a DRAM refresh cycle, for example). When the processor finally is able to gain access to the RAM array, it may have already dropped its $\overline{LOCK}$ signal, thus allowing the dual port controller to give the other port access to the RAM array instead. An example circuit which can be used to hold $\overline{LOCK}$ active until a RDY has been received by the 80186 is shown in Figure 15.

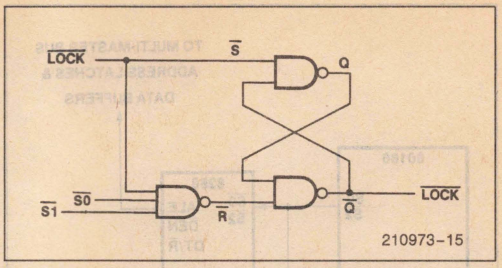


Figure 15. Circuit Holding $\overline{Lock}$ Active until Ready is Returned

3.1.7 HALT TIMING

A HALT bus cycle is used to signal the world that the 80186 CPU has executed a HLT instruction. It differs from a normal bus cycle in two important ways.

The first way in which a HALT bus cycle differs from a normal bus cycle is that since the processor is entering a halted state, none of the control lines ($\overline{RD}$ or $\overline{WR}$) will be driven active. Address and data information will not be driven by the processor, and no data will be returned. The second way a HALT bus cycle differs from a normal bus cycle is that the $\overline{S0}$–$\overline{S2}$ status lines go to their passive state (all high) during T_2 of the bus cycle, well before they go to their passive state during a normal bus cycle.

Like a normal bus cycle, however, ALE is driven active. Since no valid address information is present, the information strobed into the address latches should be ignored. This ALE pulse can be used, however, to latch the HALT status from the $\overline{S0}$–$\overline{S2}$ status lines.

The processor being halted does not interfere with the operation of any of the 80186 integrated peripheral units. This means that if a DMA transfer is pending while the processor is halted, the bus cycles associated with the DMA transfer will run. In fact, DMA latency time will improve while the processor is halted because the DMA unit will not be contending with the processor for access to the 80186 (see section 4.4.1).

3.1.8 8288 AND 8289 INTERFACING

The 8288 and 8289 are the bus controller and multimaster bus arbitration devices used with the 8086 and 8088. Because the 80186 bus is similar to the 8086 bus, they can be directly used with the 80186. Figure 16 shows an 80186 interconnection to these two devices.

The 8288 bus controller generates control signals ($\overline{RD}$, $\overline{WR}$, ALE, DT/$\overline{R}$, $\overline{DEN}$, etc.) for an 8086 maximum mode system. It derives its information by decoding status lines $\overline{S0}$–$\overline{S2}$ of the processor. Because the 80186 and the 8086 drive the same status information on these lines, the 80186 can be directly connected to the 8288 just as in an 8086 system. Using the 8288 with the 80186 does not prevent using the 80186 control signals directly. Many systems require both local bus control signals and system bus control signals. In this type of system, the 80186 lines could be used as the local sig-

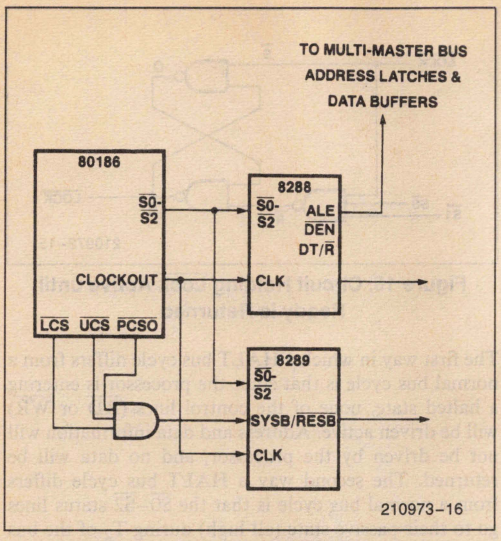


Figure 16. 80186/8288/8289 Interconnection

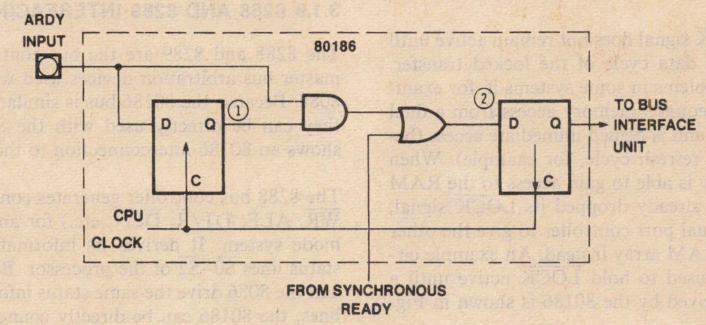
nals, with the 8288 lines used as the system signals. Note that in an 80186 system, the 8288 generated ALE pulse occurs later than that of the 80186 itself. In many multimaster bus systems, the 8288 ALE pulse should be used to strobe the addresses into the system bus address latches to insure that the address hold times are met.

The 8289 bus arbiter arbitrates the use of a multi-master system bus among various devices each of which can become the bus master. This component also decodes status lines $\overline{S0}$–$\overline{S2}$ of the processor directly to determine when the system bus is required. When the system bus is required, the 8289 forces the processor to wait until it has acquired control of the bus, then it allows the processor to drive address, data and control information onto the system bus. The system determines when it requires system bus resources by an address decode. Whenever the address being driven coincides with the address of an on-board resource, the system bus is not required and thus will not be requested. The circuit shown factors the 80186 chip select lines to determine when the system bus should be requested, or when the 80186 request can be satisfied using a local resource.

3.1.9 READY INTERFACING

The 80186 provides two ready lines, a synchronous ready (SRDY) line and an asynchronous ready (ARDY) line. These lines signal the processor to insert wait states (T_W) into a CPU bus cycle. This allows slower devices to respond to CPU service requests (reads or writes). Wait states will only be inserted when both ARDY and SRDY are low, i.e., only one of ARDY or SRDY need be active to terminate a bus cycle. Any number of wait states may be inserted into a bus cycle. The 80186 will ignore the RDY inputs during any accesses to the integrated peripheral registers, and to any area where the chip select ready bits indicate that the external ready should be ignored.

The timing required by the two RDY lines is different. The ARDY line is meant to be used with asynchronous ready inputs. Thus, inputs to this line will be internally synchronized to the CPU clock before being presented to the processor. The synchronization circuitry used with the ARDY line is shown in Figure 17. Figure 18A and 18B show valid and invalid transitions of the ARDY line (and subsequent wait state insertion). The first flip-flop is used to "resolve" the asynchronous transition of the ARDY line. It will achieve a definite level (either high or low) before its output is latched



NOTES:
1. Asynchronous Resolution Flip Flop
2. Ready Latch Flip Flop

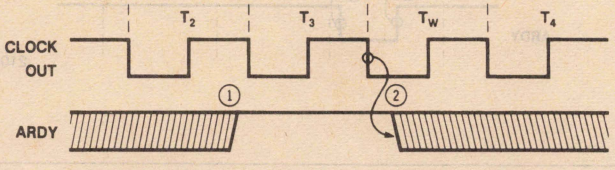
The second flip-flop is not actually in the circuit. It is drawn here only to show the functional equivalent of the interface to the BIU.

Figure 17. Asynchronous Ready Circuitry of the 80186

into the second flip-flop for presentation to the CPU. When latched high, it allows the level present on the ARDY line to pass directly to the CPU; when latched low, it forces not ready to be presented to the CPU (See Appendix B for 80186 synchronizer information).

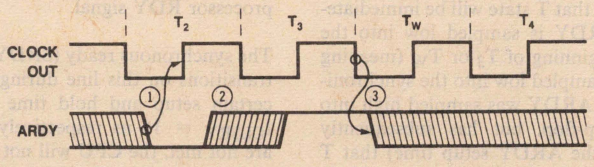
With this scheme, notice that only the active going edge of the ARDY signal is synchronized. Once the synchronization flip-flop has sampled high, the ARDY input directly drives the RDY flip-flop. Since inputs to this RDY flip-flop must satisfy certain setup and hold times, it is important that these setup and hold times (t_{ARYLCL} = 35 ns and t_{CHARYX} = 15 ns respectively) be satisfied by any inactive going transition of the ARDY line. The reason ARDY is implemented in this manner is to allow a slow device the greatest amount of time to respond with a not ready after it has been selected. In a normally ready system, a slow device must respond with a not ready quickly after it has been selected to prevent the processor from continuing and accessing invalid data from the slow device. By implementing ARDY in the above fashion, the slow device has an additional clock phase to respond with a not ready.

If RDY is sampled active into the RDY flip-flop at the beginning of T_3 or T_W (meaning that ARDY was sam-



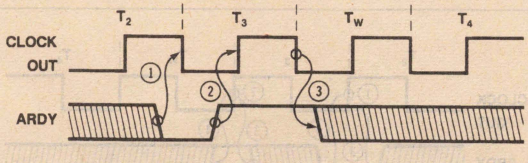
210973-18

NOTES:
1. No set-up or hold times required
2. t_{CLARYX}: Clock low to ARDY inactive (ARDY active hold time) = 15 ns min



210973-19

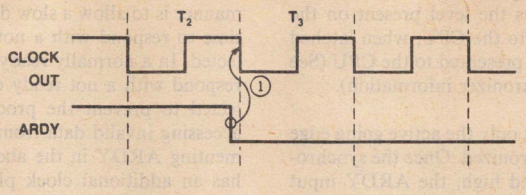
NOTES:
1. t_{ARYHCH}: ARDY valid until clock high (ARDY inactive set-up time to clock high) = 20 ns min
2. No set-up or hold time required ONLY if ① is guaranteed
3. t_{CLARYX}: Clock low to ARDY inactive (ARDY active hold time) = 15 ns min



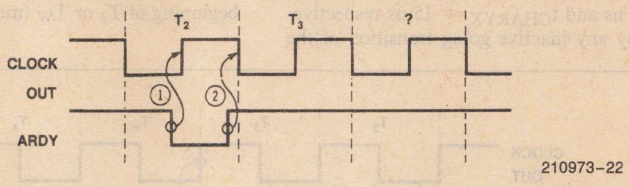
210973-20

NOTES:
1. t_{ARYLCL}: ARDY low to clock low (ARDY inactive set-up time to clock low) = 35 ns min must be satisfied since synchronizing FLIP-FLOP has sampled active
2. t_{ARYHCH}: ARDY high to clock high (ARDY active set-up time) = 20 ns min must be satisfied ONLY to guarantee recognition at the next clock (i.e., to guarantee synchronizing FLIP-FLOP will sample ARDY active)
3. t_{CLARYX}: Clock low to ARDY inactive (ARDY active hold time) = 15 ns

Figure 18A. Valid ARDY Transitions



NOTES:
1. Less than 35 ns

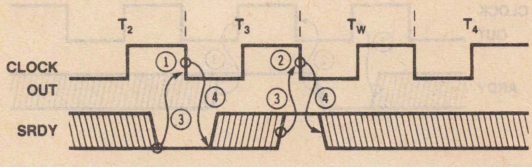


NOTES:
1. Less than 20 ns
2. Less than 35 ns

Figure 18B. Invalid ARDY Transitions

pled high into the synchronization flip-flop in the middle of a T state, and has remained high until the beginning of the next T state), that T state will be immediately followed by T_4. If RDY is sampled low into the RDY flip-flop at the beginning of T_3 or T_W (meaning that either ARDY was sampled low into the synchronization flip-flop OR that ARDY was sampled high into the synchronization flip-flop, but has subsequently changed to low before the ARDY setup time) that T state will be immediately followed by a wait state (T_W). Any asynchronous transition on the ARDY line not occurring during the above times, that is, when the processor is not "looking at" the ready lines, will not cause CPU malfunction. Again, for ARDY to force wait states to be inserted, SRDY must be driven low, since they are internally ORed together to form the processor RDY signal.

The synchronous ready (SRDY) line requires that ALL transitions on this line during T_2, T_3 or T_W satisfy a certain setup and hold time (t_{SRYCL} = 35 ns and t_{CLSRY} = 15 ns respectively). If these requirements are not met, the CPU will not function properly. Valid transitions on this line, and subsequent wait state insertion is shown in Figure 19. The processor looks at this line at the beignning of each T_3 and T_W. If the line is sampled active at the beginning of either of these two cycles, that cycle will be immediately followed by T_4.



NOTES:
1. Decision: Ready, T-State will be followed by a wait state
2. Decision: Ready, T-State will **not** be followed by a wait state
3. t_{SRYCL}: Synchronous ready stable until clock low (SRDY set-up time) = 35 ns min
4. t_{CLSRY}: Clock low until synchronous ready transition (SRDY hold time) = 15 ns min

Figure 19. Valid Transitions on the 80186

On the other hand, if the line is sampled inactive at the beginning of either of these two cycles, that cycle will be followed by a T_W. Any asynchronous transition on the SRDY line not occurring at the beginning of T_3 or T_W, that is, when the processor is not "looking at" the ready lines will not cause CPU malfunction.

3.1.10 BUS PERFORMANCE ISSUES

Bus cycles occur sequentially, but do not necessarily come immediately one after another, that is the bus may remain idle for several T states (T_i) between each bus access initiated by the 80186. This occurs whenever the 80186 internal queue is full and no read/write cycles are being requested by the execution unit or integrated DMA unit. The reader should recall that a separate unit, the bus interface unit, fetches opcodes (including immediate data) from memory, while the execution unit actually executes the pre-fetched instructions. The number of clock cycles required to execute an 80186 instruction vary from 2 clock cycles for a register to register move to 67 clock cycles for an integer divide.

If a program contains many long instructions, program execution will be CPU limited, that is, the instruction queue will be constantly filled. Thus, the execution unit does not need to wait for an instruction to be fetched. If a program contains mainly short instructions or data move instructions, the execution will be bus limited. Here, the execution unit will be required to wait often for an instruction to be fetched before it continues its operation. Programs illustrating this effect and performance degradation of each with the addition of wait states are given in appendix G.

All instruction fetches are word (16-bit) fetches from even addresses unless the fetch occurs as a result of a jump to an odd location. This maximizes the utilization of each bus cycle used for instruction fetching, since each fetch will access two bytes of information. It is also good programming practice to locate all word data at even locations, so that both bytes of the word may be accessed in a single bus cycle (see discussion on data bus interfacing for further information, section 3.1.3 of this note).

Although the amount of bus utilization, i.e., the percentage of bus time used by the 80186 for instruction fetching and execution required for top performance will vary considerably from one program to another, a typical instruction mix on the 80186 will require greater bus utilization than the 8086. This is caused by the higher performance execution unit requiring instructions from the prefetch queue at a greater rate. This also means that the effect of wait states is more pronounced in an 80186 system than in an 8086 system. In all but a few cases, however, the performance degradation incurred by adding a wait state is less than might be expected because instruction fetching and execution are performed by separate units.

3.2 Example Memory Systems

3.2.1 2764 INTERFACE

With the above knowledge of the 80186 bus, various memory interfaces may be generated. One of the simplest of these is the example EPROM interface shown in Figure 20.

The addresses are latched using the address generation circuit shown earlier. Note that the A0 line of each EPROM is connected to the A1 address line from the 80186, NOT the A0 line. Remember, A0 only signals a data transfer on the lower 8 bits of the 16-bit data bus! The EPROM outputs are connected directly to the address/data inputs of the 80186, and the 80186 $\overline{RD}$ signal is used as the $\overline{OE}$ for the EPROMs.

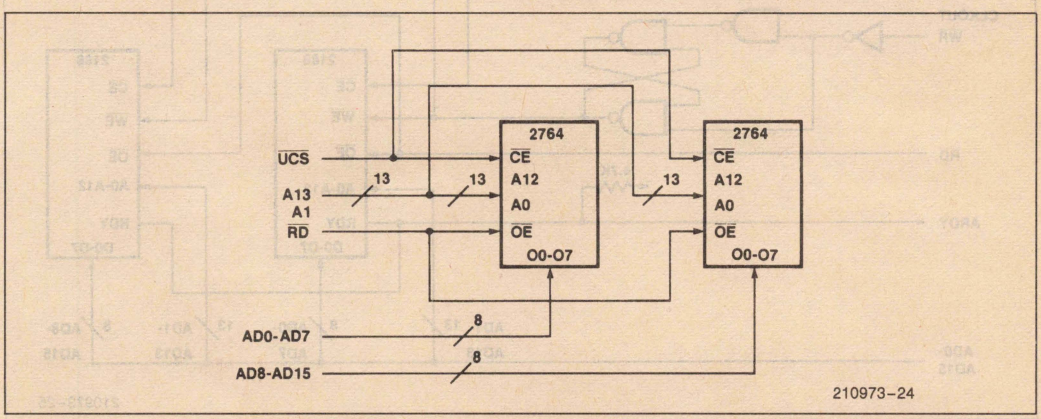


Figure 20. Example 2764/80186 Interface

The chip enable of the EPROM is driven directly by the chip select output of the 80186 (see section 8). In this configuration, the access time calculation for the EPROMs are:

time from

address: $(3 + N) * t_{CLCL} - t_{CLAV} - t_{IVOV}$ (8282) $- t_{DVCL}$

$= 375 + (N * 125) - 44 - 30 - 20$

$= 281 + (N * 125)$ ns

time from

chip select: $(3 + N) * t_{CLCL} - t_{CLCSV} - t_{DVCL}$

$= 375 + (N * 125) - 66 - 20$

$= 289 + (N * 125)$ ns

time from

$\overline{RD}$ ($\overline{OE}$): $(2 + N)\ t_{CLCL} - t_{CLRL} - t_{DVCL}$

$= 250 + (N * 125) - 70 - 20$

$= 160 + (N * 125)$ ns

where:

t_{CLAV} = time from clock low in T_1 until addresses are valid

t_{CLCL} = clock period of processor

t_{IVOV} = time from input valid of 8282 until output valid of 8282

t_{DVCL} = 186 data valid input setup time until clock low time of T_4

t_{CLCSV} = time from clock low in T_1 until chip selects are valid

t_{CLRL} = time from clock low in T_2 until $\overline{RD}$ goes low

N = number of wait states inserted

Thus, for 0 wait state operation, 250 ns EPROMs must be used. The only significant parameter not included above is t_{RHAV}, the time from $\overline{RD}$ inactive (high) until the 80186 begins driving address information. This parameter is 85 ns, which meets the 2764-25 (250 ns speed selection) output float time of 85 ns. If slower EPROMs are used, a discrete data buffer *MUST* be inserted between the EPROM data lines and the address/data bus, since these devices may continue to drive data information on the multiplexed address/data bus when the 80186 begins to drive address information for the next bus cycle.

3.2.2 2186 INTERFACE

An example interface between the 80186 and 2186 iRAMs is shown in Figure 21. This memory component is almost an ideal match with the 80186, because of its large integration, and its not requiring address latching.

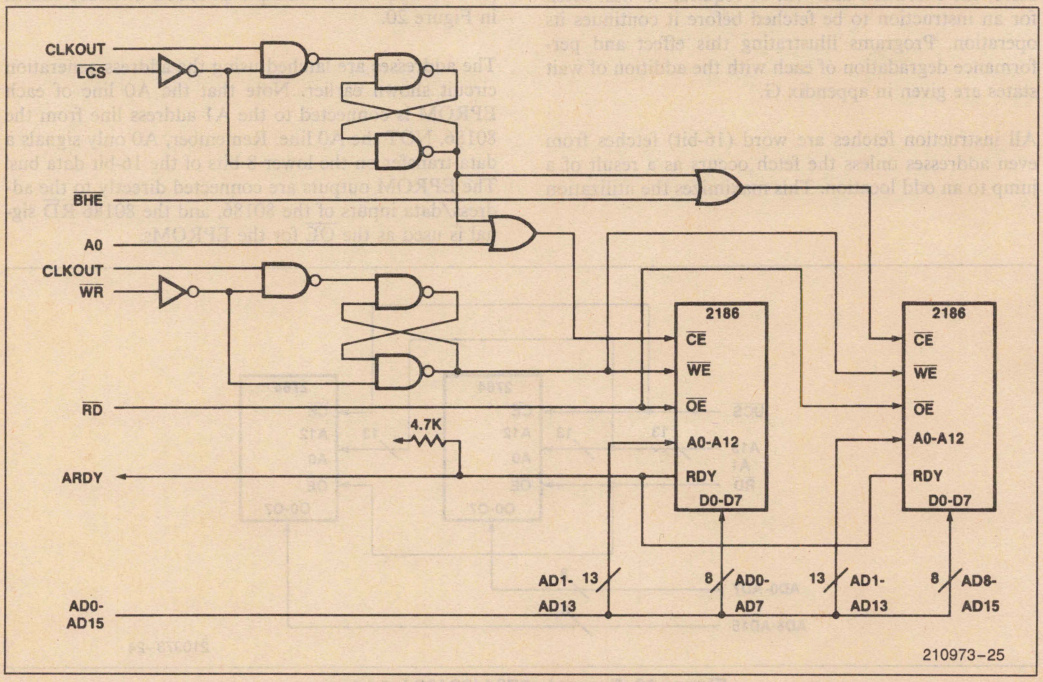


Figure 21. Example 2186/80186 Interface

The 2186 internally is a dynamic RAM integrated with refresh and control circuitry. It operates in two modes, pulse mode and late cycle mode. Pulse mode is entered if the $\overline{CE}$ signal is low to the device a maximum of 130 ns, and requires the command input ($\overline{RD}$ or $\overline{WE}$) to go active within 90 ns after $\overline{CE}$. Because of these requirements, interfacing the 80186 to the 2186 in pulse mode would be difficult. Instead, the late cycle mode is used. This affords a much simpler interface with no loss of performance. The iRAM automatically selects between these modes by the nature of the control signals.

The 2186 is a leading edge triggered device. This means that address and data information are strobed into the device on the active going (high to low) transition of the command signal. This requires both $\overline{CE}$ and $\overline{WR}$ be delayed until the address and data driven by the 80186 are guaranteed stable. Figure 21 shows a simple circuit which can be used to perform this function. Note that ALE *CANNOT* be used to delay $\overline{CE}$ if addresses are not latched externally, because this would violate the address hold time required by the 2186 (30 ns).

Because the 2186s are RAMs, data bus enables ($\overline{BHE}$ and A0, see previous section) MUST be used to factor either the chip enables or write enables of the lower and upper bytes of the 16-bit RAM memory system. If this is not done, all memory writes, including single byte writes, will write to both the upper and lower bytes of the memory system. The example system shown uses $\overline{BHE}$ and A0 as factors to the 2186 $\overline{CE}$. This may be done, because both of these signals (A0 and $\overline{BHE}$) are valid when the address information is valid from the 80186.

The 2186 requires a certain amount of recovery time between its chip enable going inactive and its chip enable going active insure proper operation. For a "normal" cycle (a read or write), this time is t_{EHEL} = 40 ns. This means that the 80186 chip select lines will go inactive soon enough at the end of a bus cycle to provide the required recovery time even if two consecutive accesses are made to the iRAMs. If the 2186 $\overline{CE}$ is asserted without a command signal ($\overline{WE}$ or $\overline{OE}$), a "False Memory Cycle" (FMC) will be generated. Whenever a FMC is generated, the recovery time is much longer; another memory cycle must not be initiated for 200 ns. As a result, if the memory system will generate FMCs, $\overline{CE}$ must be taken away in the middle of the T state (T_3 or T_W) immediately preceding T_4 to insure two consecutive cycles to the iRAM will not violate this parameter. Status going passive (all high) can be used for this purpose. These lines will all go high during the first phase of the next to last T state (either T_3 or T_W) of a bus cycle (see section 3.1.5).

Finally, since it is a dynamic device, the 2186 requires refresh cycles to maintain data integrity. The circuitry to generate these refresh cycles is integrated within the 2186. Because of this, the 2186 has a ready line which is used to suspend processor operation if a processor RAM access coincides with an internally generated refresh cycle. This is an open collector output, allowing many of them to be wire-OR'ed together, since more than one device may be accessed at a time. These lines are also normally ready, which means that they will be high whenever the 2186 is not being accessed, i.e., they will only be driven low if a processor request coincides with an internal refresh cycle. Thus, the ready lines from the iRAM must be factored into the 80186 RDY circuit only during accesses to the iRAM itself. Since the 2186 refresh logic operates asynchronously to the 80186, this RDY line must be synchronized for proper operation with the 80186, either by the integrated ready synchronizer or by an external circuit. The example circuit uses the integrated synchronizer associated with the ARDY processor input.

The ready lines of the 2186 are active unless a processor access coincides with an internal refresh cycle. These lines must go inactive soon enough after a cycle is requested to insert wait states into the data cycle. The 2186 will drive this line low within 50 ns after $\overline{CE}$ is received, which is early enough to force the 80186 to insert wait states if they are required. The primary concern here is that the ARDY line be driven not active before its setup time in the middle of T_2. This is required by the nature of the asynchronous ready synchronization circuitry of the 80186. Since the RDY pulse of the 2186 may be as narrow as 50 ns, if ready was returned after the first stage of the synchronizer, and subsequently changed state within the ready setup and hold time of the high to low going edge of the CPU clock at the end of T_2, improper operation may occur (see section 3.1.6).

The example interface shown has a zero wait state RAM read access time from $\overline{CE}$ of:

$$3 * t_{CLCL} - t_{CLCSV} - (\text{TTL delay}) - t_{DVCL}$$

$$= 375 - 66 - 30 - 20 \text{ ns}$$

$$= 259 \text{ ns}$$

where:

t_{CLCL} = CPU clock cycle time

t_{CLCSV} = time from clock low in T_1 until chip selects are valid

t_{DVCL} = 80186 data in setup time before clock low in T_4

The data valid delay from $\overline{OE}$ active is less than 100 ns, and is therefore not an access time limiter in this interface. Additionally, the 2186 data float time from $\overline{RD}$ inactive is less than the 85 ns 80186 imposed maximum. The $\overline{CE}$ generation circuit shown in Figure 21 provides an address setup time of at least 11 ns, and an address hold time of at least 35 ns (assuming a maximum two level TTL delay of less than 30 ns).

Write cycle address setup and hold times are identical to the read cycle times. The circuit shown provides at least 11 ns, write data setup and 100 ns data hold time from $\overline{WE}$, easily meeting the 0 ns setup and 40 ns hold times required by the 2186.

For more information concerning 2186 timing and interfacing, please consult the 2186 data sheet, or the application note AP-132, "Designing Memory Systems with the 8Kx8 iRAM" by John Fallin and William Righter (June 1982).

3.2.3 8203 DRAM INTERFACE

An example 8203/DRAM interface is shown in Figure 22. The 8203 provides all required DRAM control signals, address multiplexing, and refresh generation. In this circuit, the 8203 is configured to interface with 64K DRAMs.

All 8203 cycles are generated off control signals ($\overline{RD}$ and $\overline{WR}$) provided by the 80186. These signals will not go active until T_2 of the bus cycle. In addition, since the 8203 clock (generated by the internal crystal oscillator of the 8203) is asynchronous to the 80186 clock, all memory requests by the 80186 must be synchronized to the 8203 before the cycle will be run. To minimize this synchronization time, the 8203 should be used with the highest speed crystal that will maintain DRAM compatability. Even if a 25 MHz crystal is used (the maximum allowed by the 8203) two wait states will be required by the example circuit when using 150 ns DRAMs with an 8 MHz 80186, three wait states if 200 ns DRAMs are used (see timing analysis, Figure 23).

The entire RAM array controlled by the 8203 can be selected by one or a group of the 80186 provided chip selects. These chip selects can also be used to insert the wait states required by the interface.

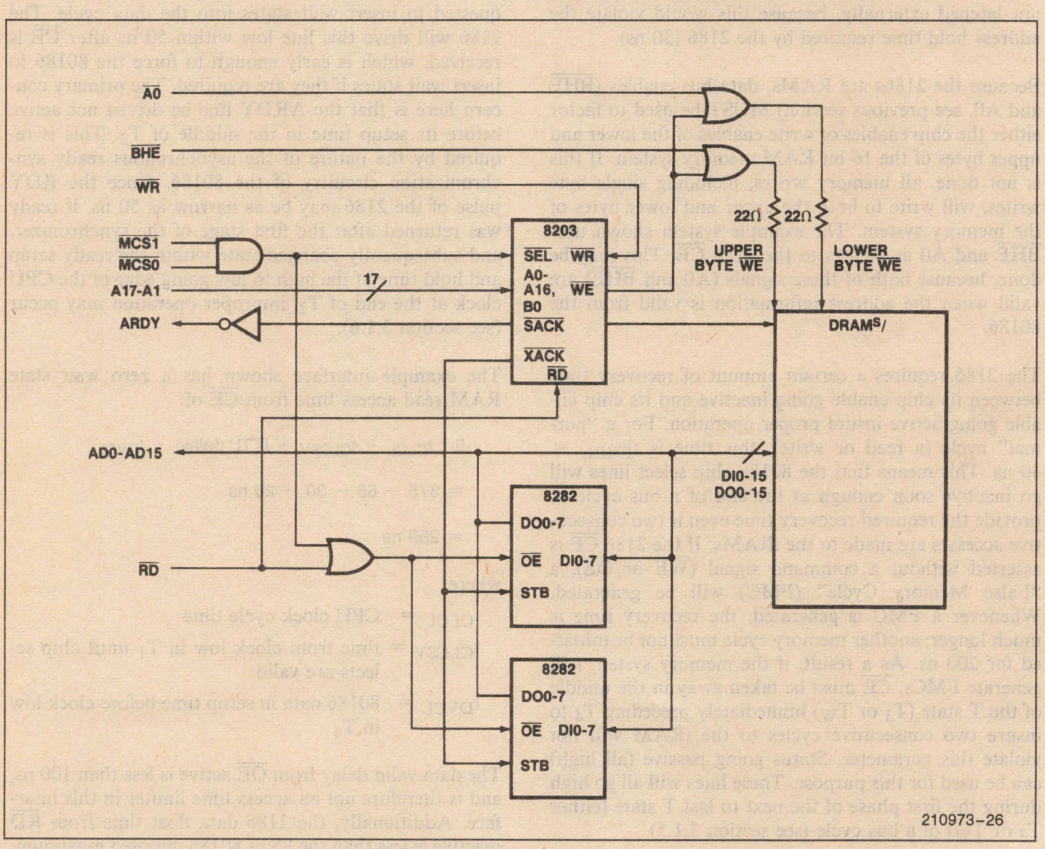
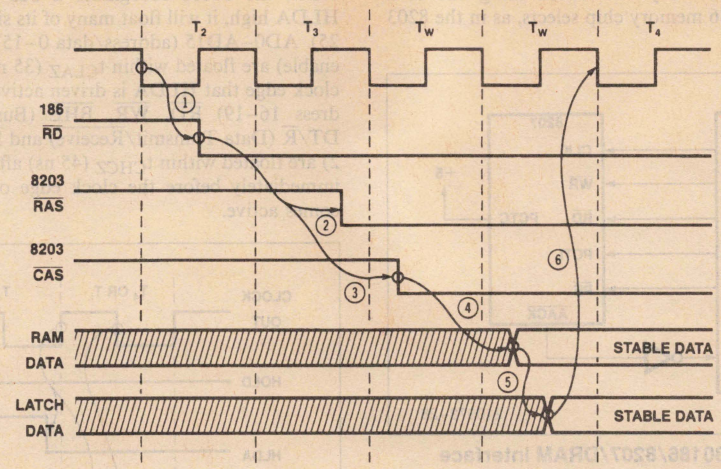


Figure 22. Example 8203/DRAM/80186 Interface



NOTES:

1. t_{CLEL}: Clock low until read low = 70 ns max
2. t_{CR}: Command active until RAS = 150 ns max*
3. t_{CC}: Command active until CAS = 245 ns max*
4. t_{CAC}: Access time from CAS = 85 ns max
5. t_{ISOU}: Input to output delay = 30 ns max
6. t_{DVCL}: Data valid to clock low (data in set up) = 20 ns min

Total Access Time = 70 + 245 + 85 + 30 + 20 = 450 ns (3.6 T-states)

① & ② are 186 specs
② & ③ are 8203 specs
④ is a 2164A-15 spec
⑤ is on 8282 spec

*Assumes 25 MHz 8203 operation

Figure 23. Example 8203/2164A-15 Access Time Calculation

Since the 8203 is operating asynchronously to the 80186, the RDY output of the 8203 (used to suspend processor operation when a processor DRAM request coincides with a DRAM refresh cycle) must be synchronized to the 80186. The 80186 ARDY line is used to provide the necessary ready synchronization. The 8203 ready outputs operate in a normally not ready mode, that is, they are only driven active when an 8203 cycle is being executed, and a refresh cycle is not being run. This is fundamentally different than the normally ready mode used by the 2186 iRAMs (see previous section). The 8203 $\overline{SACK}$ is presented to the 80186 only when the DRAM is being accessed. Notice that the $\overline{SACK}$ output of the 8203 is used, rather than the $\overline{XACK}$ output. Since the 80186 will insert at least one full CPU clock cycle between the time RDY is sampled active, and the time data must be present on the data bus, using the $\overline{XACK}$ signal would insert unnecessary additional wait states, since it does not indicate ready until valid data is available from the memory.

For more information about 8203/DRAM interfacing and timing, please consult the 8203 data sheet, or AP97A, "Interfacing Dynamic RAM to iAPX86/88 Systems Using the Intel 8202A and 8203" by Brad May (April 1982).

3.2.4 8207 DRAM INTERFACE

The 8207 advanced dual-port DRAM controller provides a high performance DRAM memory interface specifically for 80186 and 80286 microcomputer systems. This controller provides all address multiplexing and DRAM refresh circuitry. In addition, it synchronizes and arbitrates memory requests from two different ports (e.g., an 80186 and a Multibus), allowing the two ports to share memory. Finally, the 8207 provides a simple interface to the 8206 error detection and correction chip.

The simplest 8207 (and also the highest performance) interface is shown in Figure 24. This shows the 80186 connected to an 8207 using the 8207 slow cycle, synchronous status interface. In this mode, the 8207 decodes the type of cycle to be run directly from the status lines of the 80186. In addition, since the 8207 CLOCKIN is driven by the CLOCKOUT of the 80186, any performance degradation caused by required memory request synchronization between the 80186 and the 8207 is not present. Finally, the entire memory array

driven by the 8207 may be selected using one or a group of the 80186 memory chip selects, as in the 8203 interface above.

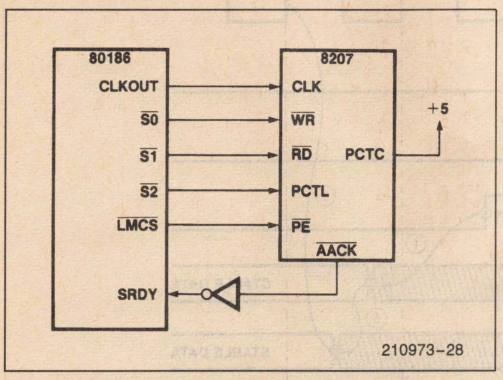


Figure 24. 80186/8207/DRAM Interface

The 8207 $\overline{AACK}$ signal may be used to generate a synchronous ready signal to the 80186 in the above interface. Since dynamic memory periodically requires refreshing, 80186 access cycles may occur simultaneously with an 8207 generated refresh cycle. When this occurs, the 8207 will hold the $\overline{AACK}$ line high until the processor initiated access is run (note, the sense of this line is reversed with respect to the 80186 SRDY input). This signal should be factored with the DRAM (8207) select input and used to drive the SRDY line of the 80186. Remember that only one of SRDY and ARDY needs to be active for a bus cycle to be terminated. If asynchronous devices (e.g., a Multibus interface) are connected to the ARDY line with the 8207 connected to the SRDY line, care must be taken in design of the ready circuit such that only one of the RDY lines is driven active at a time to prevent premature termination of the bus cycle.

3.3 HOLD/HLDA Interface

The 80186 employs a HOLD/HLDA bus exchange protocol. This protocol allows other asynchronous bus master devices (i.e., ones which drive address, data, and control information on the bus) to gain control of the bus to perform bus cycles (memory or I/O reads or writes).

3.3.1 HOLD RESPONSE

In the HOLD/HLDA protocol, a device requiring bus control (e.g., an external DMA device) raises the HOLD line. In response to this HOLD request, the 80186 will raise its HLDA line after it has finished its current bus activity. When the external device is finished with the bus, it drops its bus HOLD request. The 80186 responds by dropping its HLDA line and resuming bus operation.

When the 80186 recognizes a bus hold by driving HLDA high, it will float many of its signals (see Figure 25). AD0–AD15 (address/data 0–15) and $\overline{DEN}$ (data enable) are floated within t_{CLAZ} (35 ns) after the same clock edge that HLDA is driven active. A16–A19 (address 16–19) $\overline{RD}$, $\overline{WR}$, $\overline{BHE}$ (Bus High Enable), DT/$\overline{R}$ (Data Transmit/Receive) and $\overline{S0}$–$\overline{S2}$ (status 0–2) are floated within t_{CHCZ} (45 ns) after the clock edge immediately before the clock edge on which HLDA comes active.

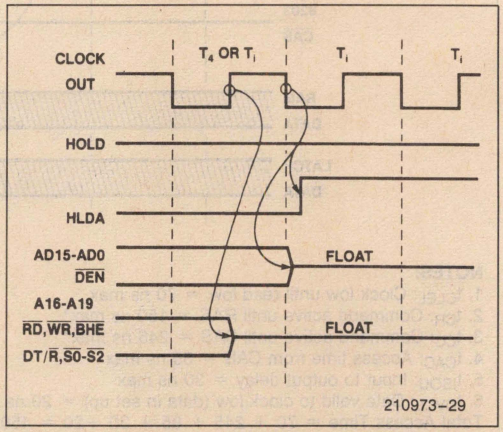


Figure 25. Signal Float/HLDA Timing of the 80186

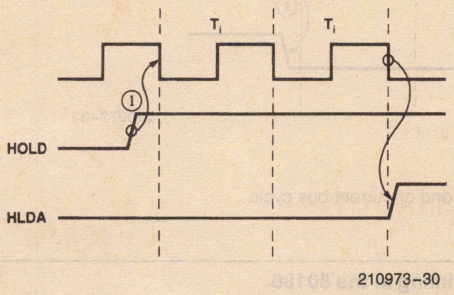
Only the above mentioned signals are floated during bus HOLD. Of the signals not floated by the 80186, some have to do with peripheral functionality (e.g., TmrOut). Many others either directly or indirectly control bus devices. These signals are ALE (Address Latch Enable, see section 3.1.2) and all the chip select lines ($\overline{UCS}$, $\overline{LCS}$, $\overline{MCS0}$–3, and $\overline{PCS0}$–6). The designer must be aware that the chip select circuitry does not look at externally generated addresses (see section 10 for a discussion of the chip select logic). Thus, for memory or peripheral devices which are addressed by external bus master devices, discrete chip select and ready generation logic must be used.

3.3.2 HOLD/HLDA TIMING AND BUS LATENCY

The time required between HOLD going active and the 80186 driving HLDA active is known as bus latency. Many factors affect this latency, including synchronization delays, bus cycle times, locked transfer times and interrupt acknowledge cycles.

The HOLD request line is internally synchronized by the 80186, and may therefore be an asynchronous signal. To guarantee recognition on a certain clock edge, it must satisfy a certain setup and hold time to the falling

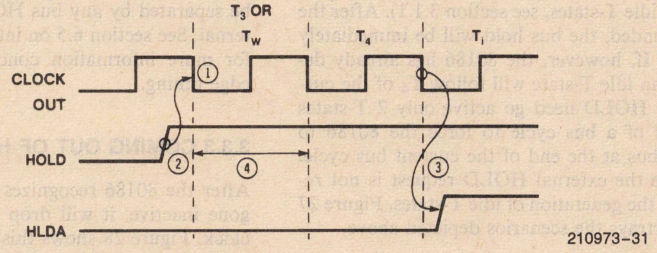
edge of the CPU clock. A full CPU clock cycle is required for this synchronization, that is, the internal HOLD signal is not presented to the internal bus arbitration circuitry until one full clock cycle after it is latched from the HOLD input (see Appendix B for a discussion of 80186 synchronizers). If the bus is idle, HLDA will follow HOLD by two CPU clock cycles plus a small amount of setup and propagation delay time. The first clock cycle synchronizes the input; the second signals the internal circuitry to initiate a bus hold. (See Figure 26).



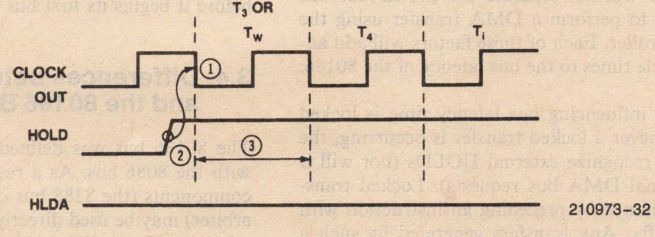
NOTES:
1. t_{HVCL}: Hold valid until clock low = 25 ns min
2. t_{CLHAV}: Clock low until HLDA active = 50 ns max

Figure 26. 80186 Idle Bus Hold/HLDA Timing

Many factors influence the number of clock cycles between a HOLD request and a HLDA. These may make bus latency longer than the best case shown above. Perhaps the most important factor is that the 80186 will not relinquish the local bus until the bus is idle. An idle bus occurs whenever the 80186 is not performing any bus transfers. As stated in section 3.1.1, when the bus is idle, the 80186 generates idle T-states. The bus can become idle only at the end of a bus cycle. Thus, the 80186 can recognize HOLD only after the end of its current bus cycle. The 80186 will normally insert no T_i states between T_4 and T_1 of the next bus cycle if it requires any bus activity (e.g., instruction fetches or I/O reads). However, the 80186 may not have an immediate need for the bus after a bus cycle, and will insert T_i states independent of the HOLD input (see section 3.1.7).

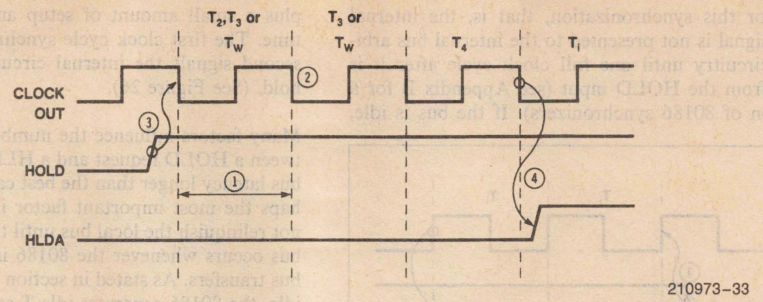


NOTES:
1. Decision: No additional internal bus cycles required, idle T-states will be inserted after T_4
2. Greater than 25 ns (t_{HVCL})
3. Less than 50 ns (t_{CLHAV})
4. HOLD request internally synchronized



NOTES:
1. Decision: Additional internal bus cycles required, no idle T-states will be inserted, Hold not active soon enough to force idle T-states
2. Greater than 25 ns (t_{HVCL}): not required since it will not get recognized anyway
3. HOLD request internally synchronized

Figure 27. HLD/HLDA Timing in the 80186



NOTES:
1. HOLD request internally synchronized
2. Decision: HOLD request active, idle t-states will be inserted at end of current bus cycle
3. Greater than 25 ns
4. Less than 50 ns

Figure 27A. HOLD/HLDA Timing in the 80186

When the HOLD request is active, the 80186 will be forced to proceed from T_4 to T_i in order that the bus may be relinquished. HOLD must go active 3 T-states before the end of a bus cycle to force the 80186 to insert idle T-states after T_4 (one to synchronize the request, and one to signal the 80186 that T_4 of the bus cycle will be followed by idle T-states, see section 3.1.1). After the bus cycle has ended, the bus hold will be immediately acknowledged. If, however, the 80186 has already determined that an idle T-state will follow T_4 of the current bus cycle, HOLD need go active only 2 T-states before the end of a bus cycle to force the 80186 to relinquish the bus at the end of the current bus cycle. This is because the external HOLD request is not required to force the generation of idle T-states. Figure 27 graphically portrays the scenarios depicted above.

An external HOLD has higher priority than both the 80186 CPU or integrated DMA unit. However, an external HOLD will not separate the two cycles needed to perform a word access when the word accessed is located at an odd location (see section 3.1.3). In addition, an external HOLD will not separate the two-to-four bus cycles required to perform a DMA transfer using the integrated controller. Each of these factors will add additional bus cycle times to the bus latency of the 80186.

Another factor influencing bus latency time is locked transfers. Whenever a locked transfer is occurring, the 80186 will not recognize external HOLDs (nor will it recognize internal DMA bus requests). Locked transfers are programmed by preceding an instruction with the LOCK prefix. Any transfers generated by such a prefixed instruction will be locked, and will not be separated by any external bus requesting device. String instructions may be locked. Since string transfers may require thousands of bus cycles, bus latency time will suffer if they are locked.

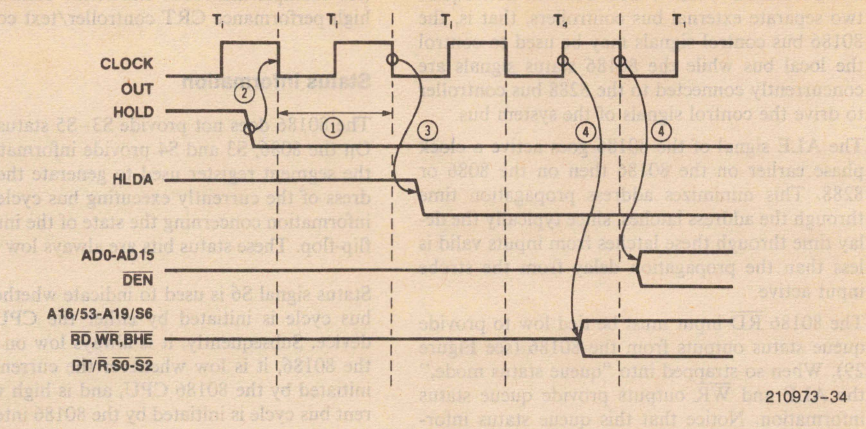
The final factor affecting bus latency time is interrupt acknowledge cycles. When an external interrupt controller is used, or if the integrated interrupt controller is used in iRMX 86 mode (see section 4.4.1) the 80186 will run two interrupt acknowledge cycles back to back. These cycles are automatically "locked" and will never be separated by any bus HOLD, either internal or external. See section 6.5 on interrupt acknowledge timing for more information concerning interrupt acknowledge timing.

3.3.3 COMING OUT OF HOLD

After the 80186 recognizes that the HOLD input has gone inactive, it will drop its HLDA line in a single clock. Figure 28 shows this timing. The 80186 will insert only two T_i after HLDA has gone inactive, assuming that the 80186 has internal bus cycles to run. During the last T_i, status information will go active concerning the bus cycle about to be run (see section 3.1.1). If the 80186 has no pending bus activity, it will maintain all lines floating (high impedance) until the last T_i before it begins its first bus cycle after the HOLD.

3.4 Differences between the 8086 Bus and the 80186 Bus

The 80186 bus was defined to be upward compatible with the 8086 bus. As a result, the 8086 bus interface components (the 8288 bus controller and the 8289 bus arbiter) may be used directly with the 80186. There are a few significant differences between the two processors which should be considered.



NOTES:
1. HOLD internally synchronized
2. Greater than 25 ns
3. Less than 50 ns
4. Lines come out of float only if a bus cycle is pending

Figure 28. 80186 Coming out of Hold

CPU Duty Cycle and Clock Generator

The 80186 employs an integrated clock generator which provides a 50% duty cycle CPU clock (1/2 of the time it is high, the other 1/2 of the time it is low). This is different than the 8086, which employs an external clock generator (the 8284A) with a 33% duty cycle CPU clock (1/3 of the time it is high, the other 2/3 of the time, it is low). These differences manifest themselves as follows:

1) No oscillator output is available from the 80186, as it is available from the 8284A clock generator.
2) The 80186 does not provide a PCLK (50% duty cycle, 1/2 CPU clock frequency) output as does the 8284A.
3) The clock low phase of the 80186 is narrower, and the clock high phase is wider than on the same speed 8086.
4) The 80186 does not internally factor AEN with RDY. This means that if both RDY inputs (ARDY and SRDY) are used, external logic must be used to prevent the RDY not connected to a certain device from being driven active during an access to this device (remember, only one RDY input needs to be active to terminate a bus cycle, see section 3.1.6).
5) The 80186 concurrently provides both a single asynchronous ready input and a single synchronous ready input, while the 8284A provides either two synchronous ready inputs or two asynchronous ready inputs as a user strapable option.
6) The CLOCKOUT (CPU clock output signal) drive capacity of the 80186 is less than the CPU clock drive capacity of the 8284A. This means that not as many high speed devices (e.g., Schottky TTL flip-flops) may be connected to this signal as can be used with the 8284A clock output.
7) The crystal or external oscillator used by the 80186 is twice the CPU clock frequency, while the crystal or external oscillator used with the 8284A is three times the CPU clock frequency.

Local Bus Controller and Control Signals

The 80186 simultaneously provides both local bus controller outputs ($\overline{RD}$, $\overline{WR}$, ALE, $\overline{DEN}$ and DT/$\overline{R}$) and status outputs ($\overline{S0}$, $\overline{S1}$ $\overline{S2}$) for use with the 8288 bus controller. This is different from the 8086 where the local bus controller outputs (generated only in min mode) are sacrificed if status outputs (generated only in max mode) are desired. These differences will manifest themselves in 8086 systems and 80186 systems as follows:

1) Because the 80186 can simultaneously provide local bus control signals and status outputs, many systems supporting both a system bus (e.g.,

a MULTIBUS®) and a local bus will not require two separate external bus controllers, that is, the 80186 bus control signals may be used to control the local bus while the 80186 status signals are concurrently connected to the 8288 bus controller to drive the control signals of the system bus.

2) The ALE signal of the 80186 goes active a clock phase earlier on the 80186 then on the 8086 or 8288. This minimizes address propagation time through the address latches, since typically the delay time through these latches from inputs valid is less than the propagation delay from the strobe input active.

3) The 80186 $\overline{RD}$ input must be tied low to provide queue status outputs from the 80186 (see Figure 29). When so strapped into "queue status mode," the ALE and $\overline{WR}$ outputs provide queue status information. Notice that this queue status information is available one clock phase earlier from the 80186 than from the 8086 (see Figure 30).

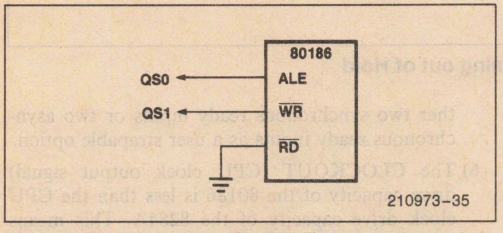


Figure 29. Generating Queue Status Information from the 80186

HOLD/HLDA vs. RQ/GT

As discussed earlier, the 80186 uses a HOLD/HLDA type of protocol for exchanging bus mastership (like the 8086 in min mode) rather than the RQ/GT protocol used by the 8086 in max mode. This allows compatibility with Intel's the new generation of high performance/high integration bus master peripheral devices (for example the 82586 Ethernet® controller or 82730 high performance CRT controller/text coprocessor).

Status Information

The 80186 does not provide S3–S5 status information. On the 8086, S3 and S4 provide information regarding the segment register used to generate the physical address of the currently executing bus cycle. S5 provides information concerning the state of the interrupt enable flip-flop. These status bits are always low on the 80186.

Status signal S6 is used to indicate whether the current bus cycle is initiated by either the CPU or a DMA device. Subsequently, it is always low on the 8086. On the 80186, it is low whenever the current bus cycle is initiated by the 80186 CPU, and is high when the current bus cycle is initiated by the 80186 integrated DMA unit.

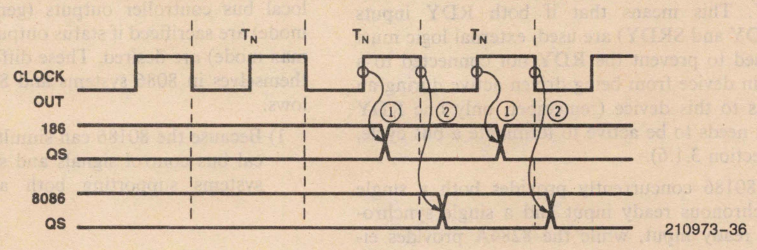
Bus Drive

The 80186 output drivers will drive 200 pF loads. This is double that of the 8086 (100 pF). This allows larger systems to be constructed without the need for bus buffers. It also means that it is very important to provide good grounds to the 80186, since its large drivers can discharge its outputs very quickly causing large current transients on the 80186 ground pins.

Misc.

The 80186 does not provide early and late write signals, as does the 8288 bus controller. The $\overline{WR}$ signal generated by the 80186 corresponds to the early write signal of the 8288. This means that data is not stable on the address/data bus when this signal is driven active.

The 80186 also does not provide differentiated I/O and memory read and write command signals. If these signals are desired, an external 8288 bus controller may be used, or the $\overline{S2}$ signal may be used to synthesize differentiated commands (see section 3.1.4).



NOTES:
1. 80186 changes queue status off falling edge of CLK
2. 8086 changes queue status off rising edge of CLK

Figure 30. 80186 and 8086 Queue Status Generation

4.0 DMA UNIT INTERFACING

The 80186 includes a DMA unit which provides two independent high speed DMA channels. These channels operate independently of the CPU, and drive all integrated bus interface components (bus controller, chip selects, etc.) exactly as the CPU (see Figure 31). This means that bus cycles initiated by the DMA unit are exactly the same as bus cycles initiated by the CPU (except that S6 = 1 during all DMA initiated cycles, see section 3.1). Thus interfacing with the DMA unit itself is very simple, since except for the addition of the DMA request connection, it is exactly the same as interfacing to the CPU.

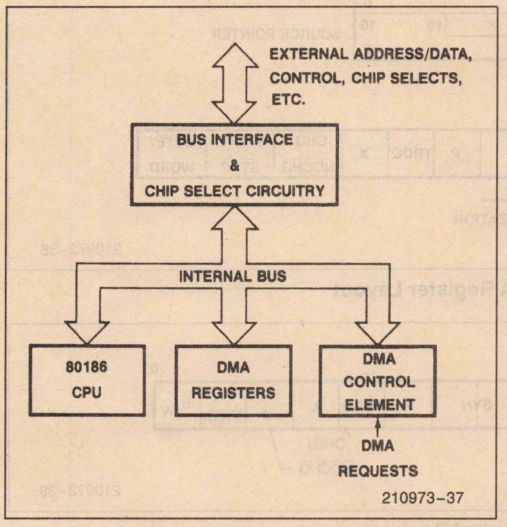


Figure 31. 80186 CPU/DMA Channel Internal Model

4.1 DMA Features

Each of the two DMA channels provides the following features:

- Independent 20-bit source and destination pointers which are used to access the I/O or memory location from which data will be fetched or to which data will be deposited
- Programmable auto-increment, auto-decrement or neither of the source and destination pointers after each DMA transfer
- Programmable termination of DMA activity after a certain number of DMA transfers
- Programmable CPU interruption at DMA termination
- Byte or word DMA transfers to or from even or odd memory or I/O addresses
- Programmable generation of DMA requests by:
  1) the source of the data
  2) the destination of the data
  3) timer 2 (see section 5)
  4) the DMA unit itself (continuous DMA requests)

4.2 DMA Unit Programming

Each of the two DMA channels contains a number of registers which are used to control channel operation. These registers are included in the 80186 integrated peripheral control block (see appendix A). These registers include the source and destination pointer registers, the transfer count register and the control register. The layout and interpretation of the bits in these registers is given in Figure 32.

The 20-bit source and destination pointers allow access to the complete 1 Mbyte address space of the 80186, and that all 20 bits are affected by the auto-increment or auto-decrement unit of the DMA (i.e., the DMA channels address the full 1 Mbyte address space of the 80186 as a flat, linear array without segments). When addressing I/O space, the upper 4 bits of the DMA pointer registers should be programmed to be 0. If they are not programmed 0, then the programmed value (greater than 64K in *I/O space*) will be driven onto the address bus (an area of I/O space not accessible to the CPU). The data transfer will occur correctly, however.

After every DMA transfer the 16-bit DMA transfer count register it is decremented by 1, whether a byte transfer or a word transfer has occurred. If the TC bit in the DMA control register is set, the DMA ST/$\overline{\text{STOP}}$ bit (see below) will be cleared when this register goes to 0, causing all DMA activity to cease. A transfer count of zero allows 65536 (2^{16}) transfers.

The DMA control register (see Figure 33) contains bits which control various channel characteristics, including for each of the data source and destination whether the pointer points to memory or I/O space, or whether the pointer will be incremented, decremented or left alone after each DMA transfer. It also contains a bit which selects between byte or word transfers. Two synchronization bits are used to determine the source of the DMA requests (see section 4.7). The TC bit determines whether DMA activity will cease after a programmed number of DMA transfers, and the INT bit is used to enable interrupts to the processor when this has occurred (note that an interrupt will not be generated to the CPU when the transfer count register reaches zero unless both the INT bit and the TC bit are set).

The control register also contains a start/stop (ST/$\overline{\text{STOP}}$) bit. This bit is used to enable DMA transfers. Whenever this bit is set, the channel is "armed,"

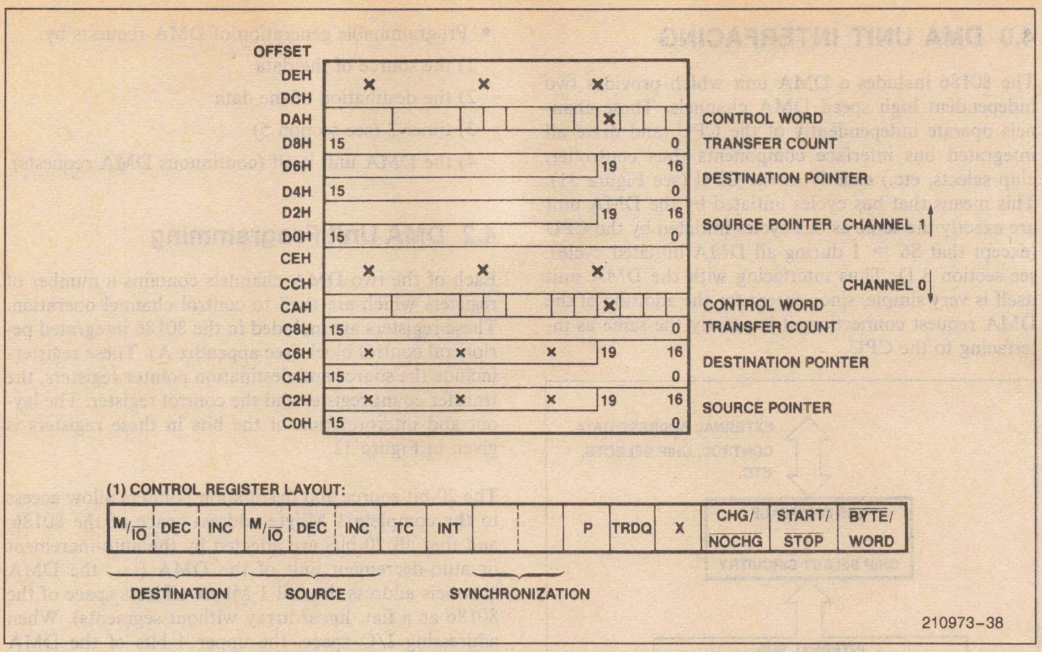


Figure 32. 80186 DMA Register Layout

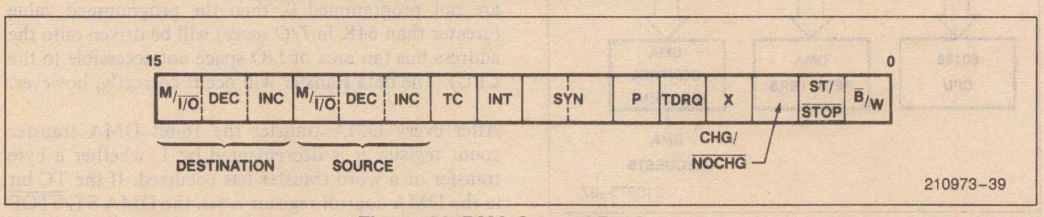
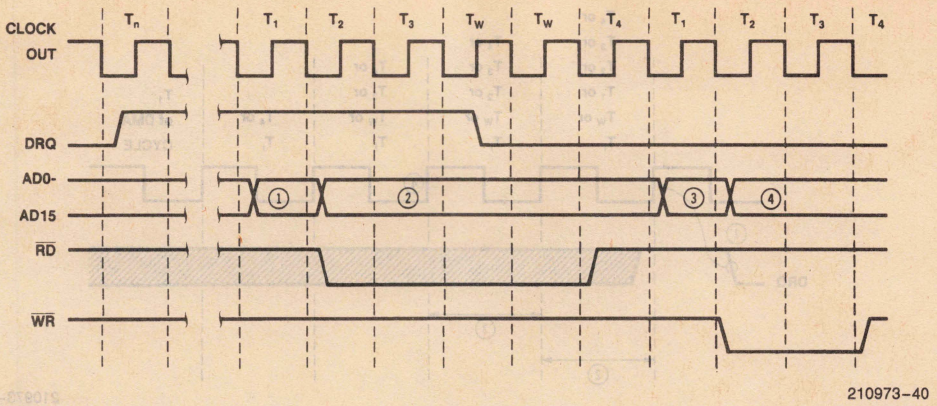


Figure 33. DMA Control Register

that is, a DMA transfer will occur whenever a DMA request is made to the channel. If this bit is cleared, no DMA transfers will be performed by the channel. A companion bit, the CHG/$\overline{\text{NOCHG}}$ bit, allows the contents of the DMA control register to be changed without modifying the state of the start/stop bit. The ST/$\overline{\text{STOP}}$ bit will only be modified if the CHG/$\overline{\text{NOCHG}}$ bit is also set during the write to the DMA control register. The CHG/$\overline{\text{NOCHG}}$ bit is write only. It will always be read back as a 0. Because DMA transfers could occur immediately after the ST/$\overline{\text{STOP}}$ bit is set, it should only be set after all other DMA controller registers have been programmed. This bit is automatically cleared when the transfer count register reaches zero and the TC bit in the DMA control register is set, or when the transfer count register reaches zero and unsynchronized DMA transfers are programmed.

All DMA unit programming registers are directly accessible by the CPU. This means the CPU can, for example, modify the DMA source pointer register after 137 DMA transfers have occurred, and have the new pointer value used for the 138th DMA transfer. If more than one register in the DMA channel is being modified at any time that a DMA request may be generated and the DMA channel is enabled (the ST/$\overline{\text{STOP}}$ bit in the control register is set), the register programming values should be placed in memory locations and moved into the DMA registers using a locked string move instruction. This will prevent a DMA transfer from occurring after only half of the register values have changed. The above also holds true if a read/modify/write type of operation is being performed (e.g., ANDing off bits in a pointer register in a single AND instruction to a pointer register mapped into memory space).



NOTES:
1. Source address
2. Source data
3. Destination address
4. Destination data

Wait states are inserted by the bus condition during the bus cycle, **not** by the DMA controller

Figure 34. Example DMA Transfer Cycle on the 80186

4.3 DMA Transfers

Every DMA transfer in the 80186 consists of two independent bus cycles, the fetch cycle and the deposit cycle (see Figure 34). During the fetch cycle, the byte or word data is accessed from memory or I/O space using the address in the source pointer register. The data accessed is placed in an internal temporary register, which is not accessible to the CPU. During the deposit cycle, the byte or word data in this internal register is placed in memory or I/O space using the address in the destination pointer register. These two bus cycles will not be separated by bus HOLD or by the other DMA channel, and one will never be run without the other except when the CPU is RESET. Notice that the bus cycles run by the DMA unit are exactly the same as memory or I/O bus cycles run by the CPU. The only difference between the two is the state of the S6 status line (which is multiplexed on the A19 line): on all CPU initiated bus cycles, this status line will be driven low; on all DMA initiated bus cycles, this status line will be driven high.

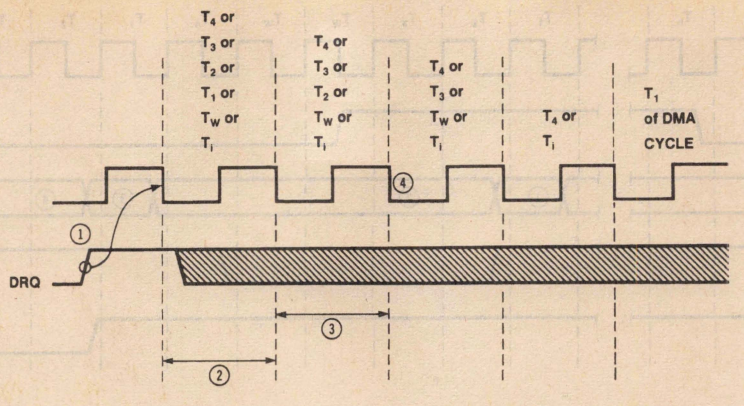
4.4 DMA Requests

Each DMA channel has a single DMA request line by which an external device may request a DMA transfer. The synchronization bits in the DMA control register determine whether this line is interpreted to be connected to the source of the DMA data or the destination of the DMA data. All transfer requests on this line are synchronized to the CPU clock before being presented to internal DMA logic. This means that any asynchronous transitions of the DMA request line will not cause the DMA channel to malfunction. In addition to external requests, DMA requests may be generated whenever the internal timer 2 times out, or continuously by programming the synchronization bits in the DMA control register to call for unsynchronized DMA transfers.

4.4.1 DMA REQUEST TIMING AND LATENCY

Before any DMA request can be generated, the 80186 internal bus must be granted to the DMA unit. A certain amount of time is required for the CPU to grant this internal bus to the DMA unit. The time between a DMA request being issued and the DMA transfer being run is known as DMA latency. Many of the issues concerning DMA latency are the same as those concerning bus latency (see section 3.3.2). The only important difference is that external HOLD always has bus priority over an internal DMA transfer. Thus, the latency time of an internal DMA cycle will suffer during an external bus HOLD.

Each DMA channel has a programmed priority relative to the other DMA channel. Both channels may be programmed to be the same priority, or one may be programmed to be of higher priority than the other channel. If both channels are active, DMA latency will suffer on the lower priority channel. If both channels are active and both channels are of the same programmed priority, DMA transfer cycles will alternate between the two channels (i.e., the first channel will perform a



210973-41

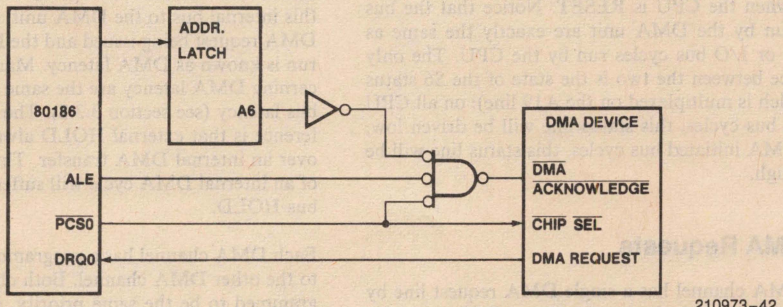
NOTES:
1. t_{DRQCL} = DMA request to clock low = 25 ns to guarantee recognition
2. Synchronizer resolution time
3. DMA unit priority arbitration, etc. time
4. Bus Interface Unit latches DMA request and decides to run DMA cycle

Figure 35. DMA Request Timing on the 80186 (showing minimum response time to request)

fetch and deposit, followed by a fetch and deposit by the second channel, etc.).

The minimum timing required to generate a DMA cycle is shown in Figure 35. Note that the minimum time from DRQ becoming active until the beginning of the first DMA cycle is 4 CPU clock cycles, that is, a DMA request is sampled 4 clock cycles before the beginning of a bus cycle to determine if any DMA activity will be required. This time is independent of the number of wait states inserted in the bus cycle. The maximum DMA latency is a function of other processor activity (see above).

Also notice that if DRQ is sampled active at 1 in Figure 35, the DMA cycle will be executed, even if the DMA request goes inactive before the beginning of the first DMA cycle. This does not mean that the DMA request is latched into the processor such that any transition on the DMA request line will cause a DMA cycle eventually. Quite the contrary, DMA request must be active at a certain time before the end of a bus cycle for the DMA request to be recognized by the processor. If the DMA request line goes inactive before that window, then no DMA cycles will be run.



210973-42

Figure 36. DMA Acknowledge Synthesis from the 80186

4.5 DMA Acknowledge

The 80186 generates no explicit DMA acknowledge signal. Instead, the 80186 performs a read or write directly to the DMA requesting device. If required, a DMA acknowledge signal can be generated by a decode of an address, or by merely using one of the $\overline{PCS}$ lines (see Figure 36). Note ALE must be used to factor the DACK because addresses are not guaranteed stable when chip selects go active. This is required because if the address is not stable when the $\overline{PCS}$ goes active, glitches can occur at the output of the DACK generation circuitry as the address lines change state. Once ALE has gone low, the addresses are guaranteed to have been stable for at least t_{AVAL} (30 ns).

4.6 Internally Generated DMA Requests

There are two types in internally synchronized DMA transfers, that is, transfer initiated by a unit integrated in the 80186. These two types are transfers in which the DMA request is generated by timer 2, or where DMA request is generated by the DMA channel itself.

The DMA channel can be programmed such that whenever timer 2 reaches its maximum count, a DMA request will be generated. This feature is selected by setting the TDRQ bit in the DMA channel control register. A DMA request generated in this manner will be latched in the DMA controller, so that once the timer request has been generated, it cannot be cleared except by running the DMA cycle or by clearing the TDRQ bits in both DMA control registers. Before any DMA requests are generated in this mode, timer 2 must be initiated and enabled.

A timer requested DMA cycle being run by either DMA channel will reset the timer request. Thus, if both channels are using it to request a DMA cycle, only one DMA channel will execute a transfer for every timeout of timer 2. Another implication of having a single bit timer DMA request latch in the DMA controller is that if another timer 2 timeout occurs before a DMA channel has a chance to run a DMA transfer, the first request will be lost, i.e., only a single DMA transfer will occur, even though the timer has timed out twice.

The DMA channel can also be programmed to provide its own DMA requests. In this mode, DMA transfer cycles will be run continuously at the maximum bus bandwidth, one after the other until the preprogrammed number of DMA transfers (in the DMA transfer count register) have occurred. This mode is selected by programming the synchronization bits in the DMA control register for unsynchronized transfers. Note that in this mode, the DMA controller will monopolize the CPU bus, i.e., the CPU will not be able to perform opcode fetching, memory operations, etc., while the DMA transfers are occurring. Also notice that the DMA will only perform the number of transfers indicated in the maximum count register regardless of the state of the TC bit in the DMA control register.

4.7 Externally Synchronized DMA Transfers

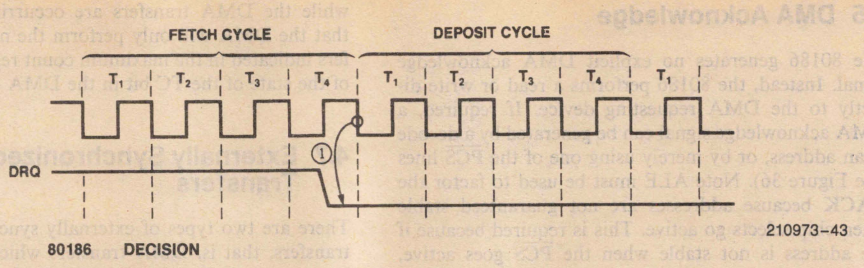
There are two types of externally synchronized DMA transfers, that is, DMA transfers which are requested by an external device rather than by integrated timer 2 or by the DMA channel itself (in unsynchronized transfers). These are source synchronized and destination synchronized transfers. These modes are selected by programming the synchronization bits in the DMA channel control register. The only difference between the two is the time at which the DMA request pin is sampled to determine if another DMA transfer is immediately required after the currently executing DMA transfer. On source synchronized transfers, this is done such that two source synchronized DMA transfers may occur one immediately after the other, while on destination synchronized transfers a certain amount of idle time is automatically inserted between two DMA transfers to allow time for the DMA requesting device to drive its DMA request inactive.

4.7.1 SOURCE SYNCHRONIZED DMA TRANSFERS

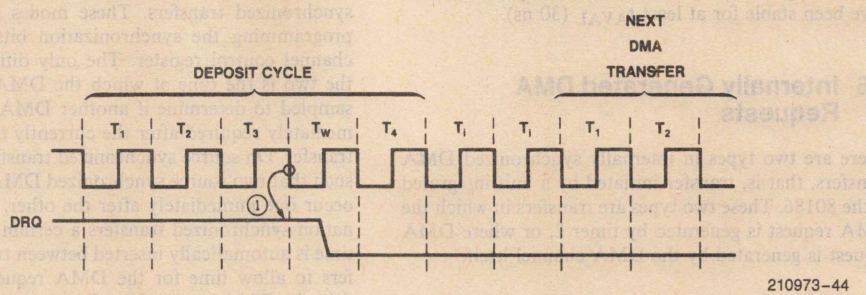
In a source synchronized DMA transfer, the source of the DMA data requests the DMA cycle. An example of this would be a floppy disk read from the disk to main memory. In this type of transfer, the device requesting the transfer is read during the fetch cycle of the DMA transfer. Since it takes 4 CPU clock cycles from the time DMA request is sampled to the time the DMA transfer is actually begun, and a bus cycle takes a minimum of 4 clock cycles, the earliest time the DMA request pin will be sampled for another DMA transfer will be at the beginning of the deposit cycle of a DMA transfer. This allows over 3 CPU clock cycles between the time the DMA requesting device receives an acknowledge to its DMA request (around the beginning of T_2 of the DMA fetch cycle), and the time it must drive this request inactive (assuming no wait states) to insure that another DMA transfer is not performed if it is not desired (see Figure 37).

4.7.2 DESTINATION SYNCHRONIZED DMA TRANSFERS

In destination synchronized DMA transfers, the destination of the DMA data requests the DMA transfer. An example of this would be a floppy disk write from main memory to the disk. In this type of transfer, the device requesting the transfer is written during the de-



NOTE:
1. Current DMA source synchronized transfer will not be immediately followed by another DMA transfer



NOTE:
1. Current DMA destination synchronized transfer will be followed immediately by another DMA transfer

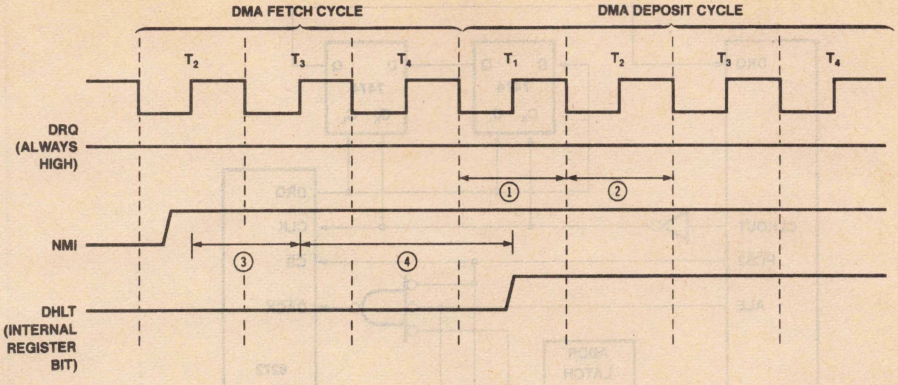
Figure 37. Source & Destination Synchronized DMA Request Timing

posit cycle of the DMA transfer. This causes a problem, since the DMA requesting device will not receive notification of the DMA cycle being run until 3 clock cycles before the end of the DMA transfer (if no wait states are being inserted into the deposit cycle of the DMA transfer) and it takes 4 clock cycles to determine whether another DMA cycle should be run immediately following the current DMA transfer. To get around this problem, the DMA unit will relinquish the CPU bus after each destination synchronized DMA transfer for at least 2 CPU clock cycles to allow the DMA requesting device time to drop its DMA request if it does not immediately desire another immediate DMA transfer. When the bus is relinquished by the DMA unit, the CPU may resume bus operation (e.g., instruction fetching, memory or I/O reads or writes, etc.). Thus, typically, a CPU initiated bus cycle will be inserted between each destination synchronized DMA transfer. If no CPU bus activity is required, however (and none can be guaranteed), the DMA unit will insert only 2 CPU clock cycles between the deposit cycle of one DMA transfer and the fetch cycle of the next DMA transfer. This means that the DMA destination requesting device must drop its DMA request at least two clock cycles before the end of the deposit cycle regardless of the number of wait states inserted into the bus cycle.

Figure 37 shows the DMA request going away too late to prevent the immediate generation of another DMA transfer. Any wait states inserted in the deposit cycle of the DMA transfer will lengthen the amount of time from the beginning of the deposit cycle to the time DMA will be sampled for another DMA transfer. Thus, if the amount of time a device requires to drop its DMA request after receiving a DMA acknowledge from the 80186 is longer than the 0 wait state 80186 maximum (100 ns), wait states can be inserted into the DMA cycle to lengthen the amount of time the device has to drop its DMA request after receiving the DMA acknowledge. Table 4 shows the amount of time between the beginning of T_2 and the time DMA request is sampled as wait states are inserted in the DMA deposit cycle.

Table 4. DMA Request Inactive Timing

Number of Wait States	Max Time (ns) For DRQ Inactive from Start of T_2
0	100
1	225
2	350
3	475



NOTES:
1. DMA request synchronization
2. Decision: Will DMA cycle be run?
   Answer: **No** DMA request is active but DHLT is set (from NMI request)
3. NMI synchronization time
4. Logic delay time from synchronized NMI until DHLT set (not: DHLT is in the interrupt control status register)

Figure 38. NMI and DMA Interaction

4.8 DMA Halt and NMI

Whenever a Non-Maskable Interrupt is received by the 80186, all DMA activity will be suspended after the end of the current DMA transfer. This is performed by the NMI automatically setting the DMA Halt (DHLT) bit in the interrupt controller status register (see section 6.3.7). The timing of NMI required to prevent a DMA cycle from occurring is shown in Figure 38. After the NMI has been serviced, the DHLT bit should be cleared by the programmer, and DMA activity will resume exactly where it left off, i.e., none of the DMA registers will have been modified. The DMA Halt bit is not automatically reset after the NMI has been serviced. It is automatically reset by the IRET instruction. This DMA halt bit may also be set by the programmer to prevent DMA activity during any critical section of code.

4.9 Example DMA Interfaces

4.9.1 8272 FLOPPY DISK INTERFACE

An example DMA interface to the 8272 Floppy Disk Controller is shown in Figure 39. This shows how a typical DMA device can be interfaced to the 80186. An example floppy disk software driver for this interface is given in Appendix C.

The data lines of the 8272 are connected, through buffers, to the 80186 AD0–AD7 lines. The buffers are required because the 8272 will not float its output drivers quickly enough to prevent contention with the 80186 driven address information after a read from the 8272 (see section 3.1.3).

DMA acknowledge for the 8272 is driven by an address decode within the region assigned to $\overline{PCS2}$. If $\overline{PCS2}$ is assigned to be active between I/O locations 0500H and 057FH, then an access to I/O location 0500H will enable only the chip select, while an access to I/O location 0501H will enable both the chip select and the DMA acknowledge. Remember, ALE must be factored into the DACK generation logic because addresses are not guaranteed stable when the chip selects become active. If ALE were not used, the DACK generation circuitry could glitch as address output changed state while the chip select was active.

Notice that the TC line of the 8272 is driven by a very similar circuit as the one generating DACK (except for the reversed sense of the output!). This line is used to terminate an 8272 command before the command has completed execution. Thus, the TC input to the 8272 is software driven in this case. Another method of driving the TC input would be to connect the DACK signal to one of the 80186 timers, and program the timer to out-

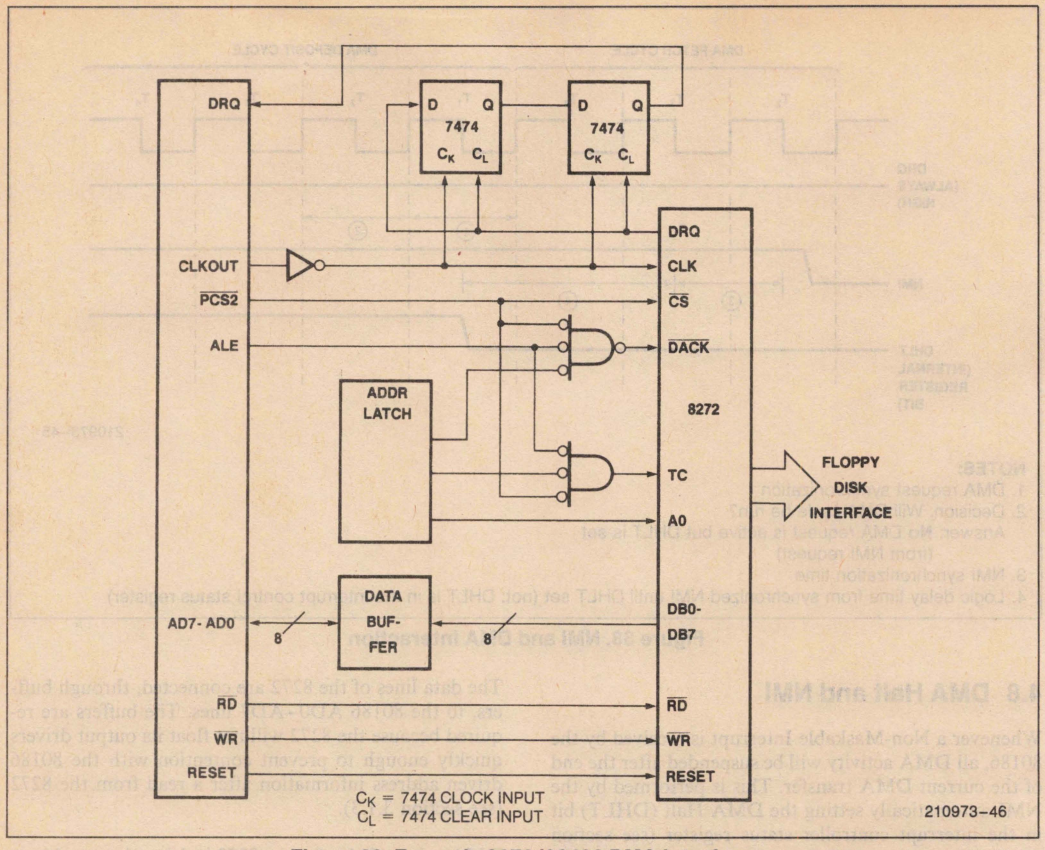


Figure 39. Example 8272/80186 DMA Interface

put a pulse to the 8272 after a certain number of DMA cycles have been run (see next section for 80186 timer information).

The above discussion assumed that a single 80186 $\overline{PCS}$ line is free to generate all 8272 select signals. If more than one chip select is free, however, different 80186 generated $\overline{PCS}$ lines could be used for each function. For example, $\overline{PCS2}$ could be used to select the 8272, $\overline{PCS3}$ could be used to drive the DACK line of the 8272, etc.

DMA requests are delayed by two clock periods in going from the 8272 to the 80186. This is required by the 8272 t_{RQR} (time from DMA request to DMA $\overline{RD}$ going active) spec of 800 ns min. This requires 6.4 80186 CPU clock cycles (at 8 MHz), well beyond the 5 minimum provided by the 80186 (4 clock cycles to the beginning of the DMA bus cycle, 5 to the beginning of T_2 of the DMA bus cycle where $\overline{RD}$ will go active). The two flip-flops add two complete CPU clock cycles to this response time.

DMA request will go away 200 ns after DACK is presented to the 8272. During a DMA write cycle (i.e., a destination synchronized transfer), this is not soon enough to prevent the immediate generation of another DMA transfer if no wait states are inserted in the deposit cycle to the 8272. Therefore, at least 1 wait state is required by this interface, regardless of the data access parameters of the 8272.

4.9.2 8274 SERIAL COMMUNICATION INTERFACE

An example 8274 synchronous/asynchronous serial chip/80186 DMA interface is shown in Figure 40. The 8274 interface is even simpler than the 8272 interface, since it does not require the generation of a DMA acknowledge signal, and the 8274 does not require the length of time between a DMA request and the DMA read or write cycle that the 8272 does. An example serial driver using the 8274 in DMA mode with the 80186 is given in Appendix C.

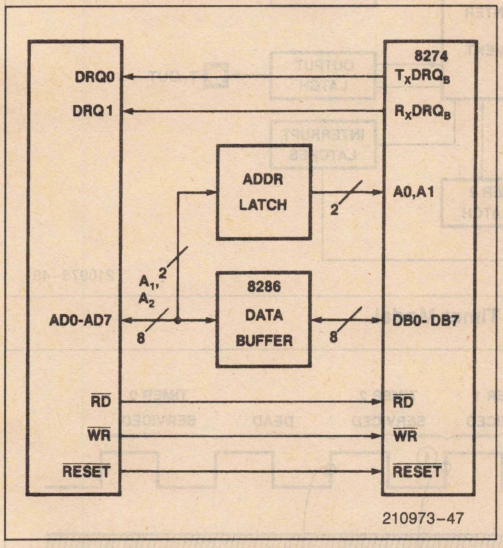


Figure 40. Example 8274/80186 DMA Interface

The data lines of the 8294 are connected through buffers to the 80186 AD0–AD7 lines. Again, these are required not because of bus drive problems, but because the 8274 will not float its drivers before the 80186 will begin driving address information on its address/data bus. If both the 8274 and the 8272 are included in the same 80186 system, they could share the same data bus buffer (as could any other peripheral devices in the system).

The 8274 does not require a DMA acknowledge signal. The first read from or write to the data register of the 8274 after the 8274 generates the DMA request signal will clear the DMA request. The time between when the control signal ($\overline{RD}$ or $\overline{WR}$) becomes active and when the 8274 will drop its DMA request during a DMA write is 150 ns, which will require at least one wait state be inserted into the DMA write cycle for proper operation of the interface.

5.0 TIMER UNIT INTERFACING

The 80186 includes a timer unit which provides three independent 16-bit timers. These timers operate independently of the CPU. Two of these have input and output pins allowing counting of external events and generation of arbitrary waveforms. The third timer can be used as a timer, as a prescaler for the other two timers, or as a DMA request source.

5.1 Timer Operation

The internal timer unit on the 80186 could be modeled by a single counter element, time multiplexed to three register banks, each of which contains different control and count values. These register banks are, in turn, dual ported between the counter element and the 80186 CPU (see Figure 41). Figure 42 shows the timer element sequencing, and the subsequent constraints on input and output signals. If the CPU modifies one of the timer registers, this change will affect the counter element the next time that register is presented to the counter element. There is no connection between the sequencing of the counter element through the timer register banks and the Bus Interface Unit's sequencing through T-states. Timer operation and bus interface operation are completely asynchronous.

5.2 Timer Registers

Each timer is controlled by a block of registers (see Figure 43). Each of these registers can be read or written whether or not the timer is operating. All processor accesses to these registers are synchronized to all counter element accesses to these registers, meaning that one will never read a count register in which only half of the bits have been modified. Because of this synchronization, one wait state is automatically inserted into any access to the timer registers. Unlike the DMA unit, locking accesses to timer registers will not prevent the timer's counter elements from accessing the timer registers.

Each timer has a 16-bit count register. This register is incremented for each timer event. A timer event can be a low-to-high transition on the external pin (for timers 0 and 1), a CPU clock transition (divided by 4 because of the counter element multiplexing), or a time out of timer 2 (for timers 0 and 1). Because the count register is 16 bits wide, up to 65536 (2^{16}) timer events can be counted by a single timer/counter. This register can be both read or written whether the timer is or is not operating.

Each timer includes a maximum count register. Whenever the timer count register is equal to the maximum count register, the count register will be reset to zero, that is, the maximum count value will never be stored in the count register. This maximum count value may

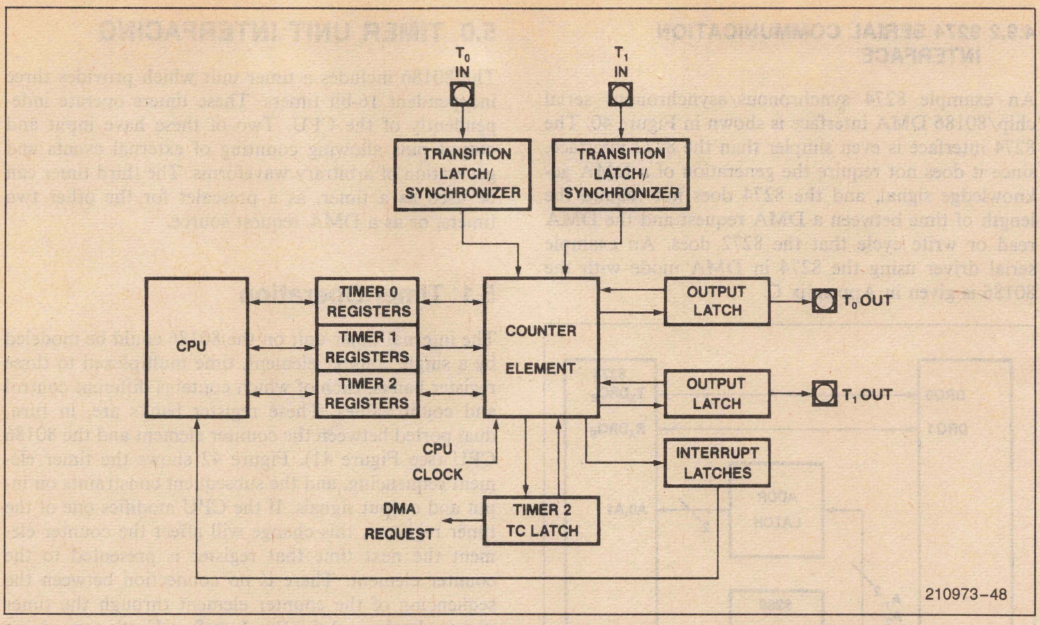
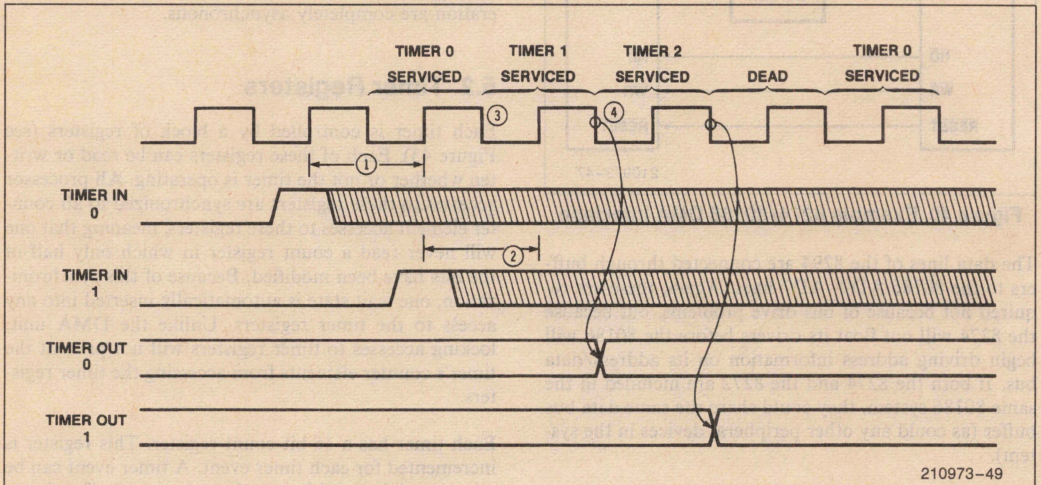


Figure 41. 80186 Timer Model



NOTES:
1. Timer in 0 resolution time
2. Timer in 1 resolution time
3. Modified count value written into 80186 timer 0 count register
4. Modified count value written into 80186 timer 1 count register

Figure 42. 80186 Counter Element Multiplexing and Timer Input Synchronization

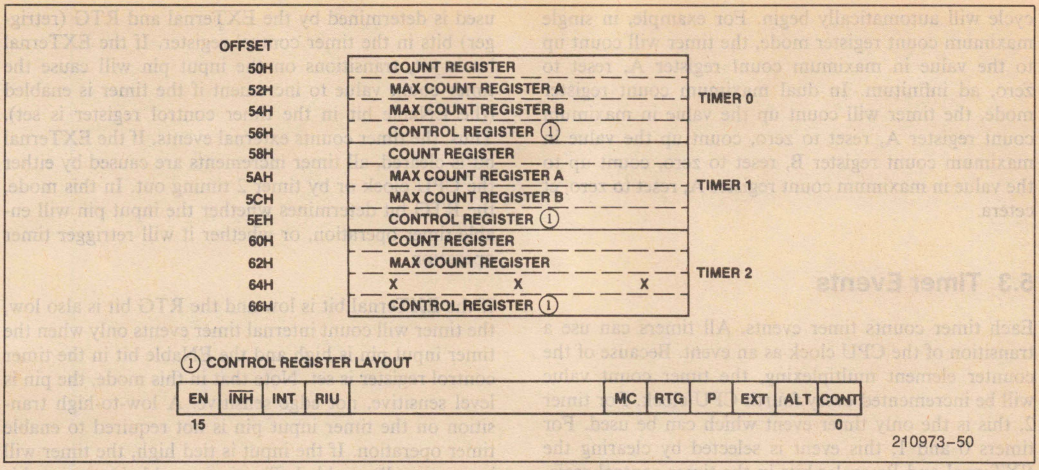


Figure 43. 80186 Timer Register Layout

be written while the timer is operating. A maximum count value of 0 implies a maximum count of 65536, a maximum count value of 1 implies a maximum count of 1, etc. The user should be aware that only equivalence between the count value and the maximum count register value is checked, that is, the count value will not be cleared if the value in the count register is greater than the value in the maximum count register. This could only occur by programmer intervention, either by setting the value in the count register greater than the value in the maximum count register, or by setting the value in the maximum count register to be less than the value in the count register. If this is programmed, the timer will count to the maximum possible count (FFFFH), increment to 0, then count up to the value in the maximum count register. The TC bit in the timer control register will not be set when the counter overflows to 0, nor will an interrupt be generated from the timer unit.

Timers 0 and 1 each contain an additional maximum count register. When both maximum count registers are used, the timer will first count up to the value in maximum count register A, reset to zero, count up to the value in maximum count register B, and reset to zero again. The ALTernate bit in the timer control register determines whether one or both maximum count registers are used. If this bit is low, only maximum count register A is used; maximum count register B is ignored. If it is high, both maximum count register A and maximum count register B are used. The RIU (register in use) bit in the timer control register indicates which maximum count register is currently being used. This bit is 0 when maximum count register A is being used, 1 when maximum count register B is being used. This RIU bit is read only. It is unaffected by any write to the timer control register. It will always be read 0 in single maximum count register mode (since only maximum count register A will be used).

Each timer can generate an interrupt whenever the timer count value reaches a maximum count value. That is, an interrupt can be generated whenever the value in maximum count register A is reached, and whenever the value in maximum count register B is reached. In addition, the MC (maximum count) bit in the timer control register is set whenever the timer count reaches a maximum count value. This bit is never automatically cleared, i.e., programmer intervention is required to clear this bit. If a timer generates a second interrupt request before the first interrupt request has been serviced, the first interrupt request to the CPU will be lost.

Each timer has an ENable bit in the timer control register. This bit is used to enable the timer to count. The timer will count timer events only when this bit is set. Any timer events occurring when this bit is reset are ignored. Any write to the timer control register will modify the ENable bit only if the INHibit bit is also set. The timer ENable bit will not be modified by a write to the timer control register if the INHibit bit is not set. The INHibit bit in the timer control register allows selective updating of the timer ENable bit. The value of the INHibit bit is not stored in a write to the timer control register; it will always be read as a 1.

Each timer has a CONTinuous bit in the timer control register. If this bit is cleared, the timer ENable bit will be automatically cleared at the end of each timing cycle. If a single maximum count register is used, the end of a timing cycle occurs when the count value resets to zero after reaching the value in maximum count register A. If dual maximum count registers are used, the end of a timing cycle occurs when the count value resets to zero after reaching the value in maximum count register B. If the CONTinuous bit is set, the ENable bit in the timer control register will never be automatically reset. Thus, after each timing cycle, another timing

cycle will automatically begin. For example, in single maximum count register mode, the timer will count up to the value in maximum count register A, reset to zero, ad infinitum. In dual maximum count register mode, the timer will count up the value in maximum count register A, reset to zero, count up the value in maximum count register B, reset to zero, count up to the value in maximum count register A, reset to zero, et cetera.

5.3 Timer Events

Each timer counts timer events. All timers can use a transition of the CPU clock as an event. Because of the counter element multiplexing, the timer count value will be incremented every fourth CPU clock. For timer 2, this is the only timer event which can be used. For timers 0 and 1, this event is selected by clearing the EXTernal and Prescaler bits in the timer control register.

Timers 0 and 1 can use timer 2 reaching its maximum count as a timer event. This is selected by clearing the EXTernal bit and setting the Prescaler bit in the timer control register. When this is done, the timer will increment whenever timer 2 resets to zero having reached its own maximum count. Note that timer 2 must be initialized and running for the other timer's value to be incremented.

Timers 0 and 1 can also be programmed to count low-to-high transitions on the external input pin. Each transition on the external pin is synchronized to the 80186 clock before it is presented to the timer circuitry, and may, therefore, be asynchronous (see Appendix B for information on 80186 synchronizers). The timer counts transitions on the input pin: the input value must go low, then go high to cause the timer increment. Any transition on this line is latched. If a transition occurs when a timer is not being serviced by the counter element, the transition on the input line will be remembered so that when the timer does get serviced, the input transition will be counted. Because of the counter element multiplexing, the maximum rate at which the timer can count is 1/4 of the CPU clock rate (2 MHz with an 8 MHz CPU clock).

5.4 Timer Input Pin Operation

Timers 0 and 1 each have individual timer input pins. All low-to-high transitions on these input pins are synchronized, latched, and presented to the counter element when the particular timer is being serviced by the counter element.

Signals on this input can affect timer operation in three different ways. The manner in which the pin signals are used is determined by the EXTernal and RTG (retrigger) bits in the timer control register. If the EXTernal bit is set, transitions on the input pin will cause the timer count value to increment if the timer is enabled (the ENable bit in the timer control register is set). Thus, the timer counts external events. If the EXTernal bit is cleared, all timer increments are caused by either the CPU clock or by timer 2 timing out. In this mode, the RTG bit determines whether the input pin will enable timer operation, or whether it will retrigger timer operation.

If the EXTernal bit is low and the RTG bit is also low, the timer will count internal timer events only when the timer input pin is high and the ENable bit in the timer control register is set. Note that in this mode, the pin is level sensitive, not edge sensitive. A low-to-high transition on the timer input pin is not required to enable timer operation. If the input is tied high, the timer will be continually enabled. The timer enable input signal is completely independent of the ENable bit in the timer control register: both must be high for the timer to count. Example uses for the timer in this mode would be a real time clock or a baud rate generator.

If the EXTernal bit is low and the RTG bit is high, the timer will act as a digital one-shot. In this mode, every low-to-high transition on the timer input pin will cause the timer to reset to zero. If the timer is enabled (i.e., the ENable bit in the timer control register is set) timer operation will begin (the timer will count CPU clock transitions or timer 2 timeouts). Timer operation will cease at the end of a timer cycle, that is, when the value in the maximum count register A is reached and the timer count value resets to zero (in single maximum count register mode, remember that the maximum count value is never stored in the timer count register) or when the value in maximum count register B is reached and the timer count value resets to zero (in dual maximum count register mode). If another low-to-high transition occurs on the input pin before the end of the timer cycle, the timer will reset to zero and begin the timing cycle again regardless of the state of the CONtinuous bit in the timer control register the RIU bit will not be changed by the input transition. If the CONtinuous bit in the timer control register is cleared, the timer ENable bit will automatically be cleared at the end of the timer cycle. This means that any additional transitions on the input pin will be ignored by the timer. If the CONtinuous bit in the timer control register is set, the timer will reset to zero and begin another timing cycle for every low-to-high transition on the input pin, regardless of whether the timer had reached the end of a timer cycle, because the timer ENable bit would not have been cleared at the end of the timing cycle. The timer will also continue counting at the end of a timer cycle, whether or not another transition has occurred on the input pin. An example use of the timer in this mode is an alarm clock time out signal or interrupt.

5.5 Timer Output Pin Operation

Timers 0 and 1 each contain a single timer output pin. This pin can perform two functions at programmer option. The first is a single pulse indicating the end of a timing cycle. The second is a level indicating the maximum count register currently being used. The timer outputs operate as outlined below whether internal or external clocking of the timer is used. If external clocking is used, however, the user should remember that the time between an external transition on the timer input pin and the time this transition is reflected in the timer out pin will vary depending on when the input transition occurs relative to the timer's being serviced by the counter element.

When the timer is in single maximum count register mode (the ALTernate bit in the timer control register is cleared) the timer output pin will go low for a single CPU clock the clock after the timer is serviced by the counter element where maximum count is reached (see Figure 44). This mode is useful when using the timer as a baud rate generator.

When the timer is programmed in dual maximum count register mode (the ALTernate bit in the timer control register is set), the timer output pin indicates which maximum count register is being used. It is low if maximum count register B is being used for the current count, high if maximum count register A is being used. If the timer is programmed in continuous mode (the CONTinuous bit in the timer control register is set), this pin could generate a waveform of any duty cycle. For example, if maximum count register A contained 10 and maximum count register B contained 20, a 33% duty cycle waveform would be generated.

5.6 Sample 80186 Timer Applications

The 80186 timers can be used for almost any application for which a discrete timer circuit would be used. These include real time clocks, baud rate generators, or event counters.

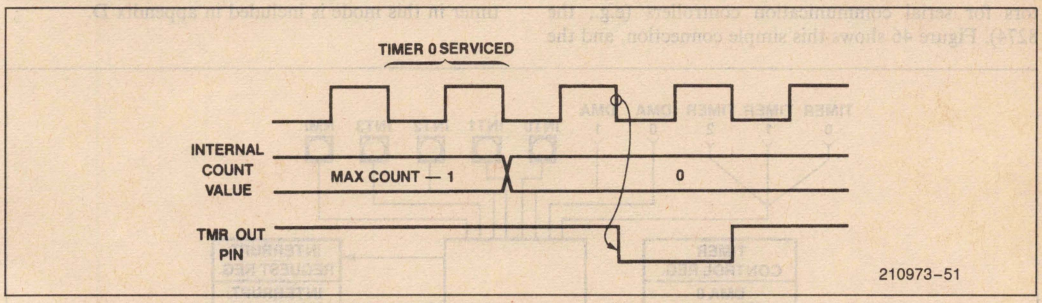


Figure 44. 80186 Timer Out Signal

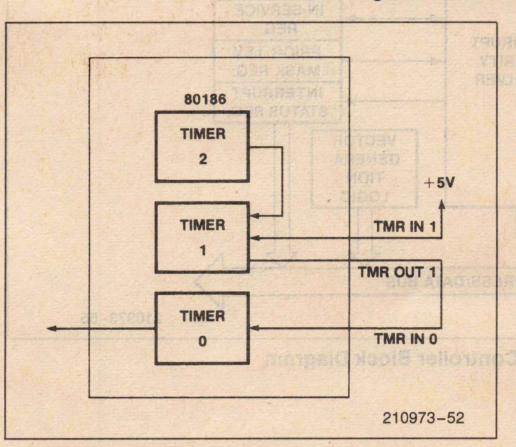


Figure 45. 80186 Real Time Clock

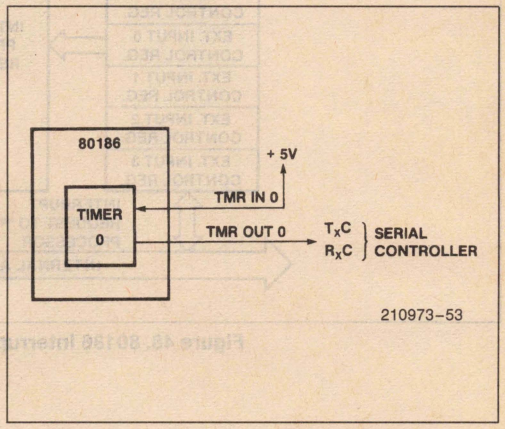


Figure 46. 80186 Baud Rate Generator

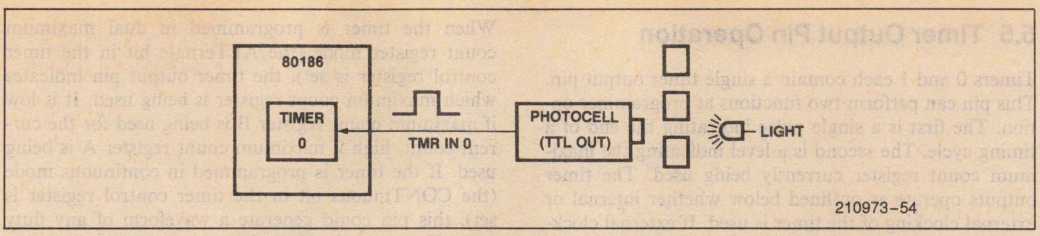


Figure 47

5.6.1 80186 TIMER REAL TIME CLOCK

The sample program in appendix D shows the 80186 timer being used with the 80186 CPU to form a real time clock. In this implementation, timer 2 is programmed to provide an interrupt to the CPU every millisecond. The CPU then increments memory based clock variables.

5.6.2 80186 TIMER BAUD RATE GENERATOR

The 80186 timers can also be used as baud rate generators for serial communication controllers (e.g., the 8274). Figure 46 shows this simple connection, and the code to program the timer as a baud rate generator is included in appendix D.

5.6.3 80186 TIMER EVENT COUNTER

The 80186 timer can be used to count events. Figure 47 shows a hypothetical set up in which the 80186 timer will count the interruptions in a light source. The number of interruptions can be read directly from the count register of the timer, since the timer counts up, i.e., each interruption in the light source will cause the timer count value to increase. The code to set up the 80186 timer in this mode is included in appendix D.

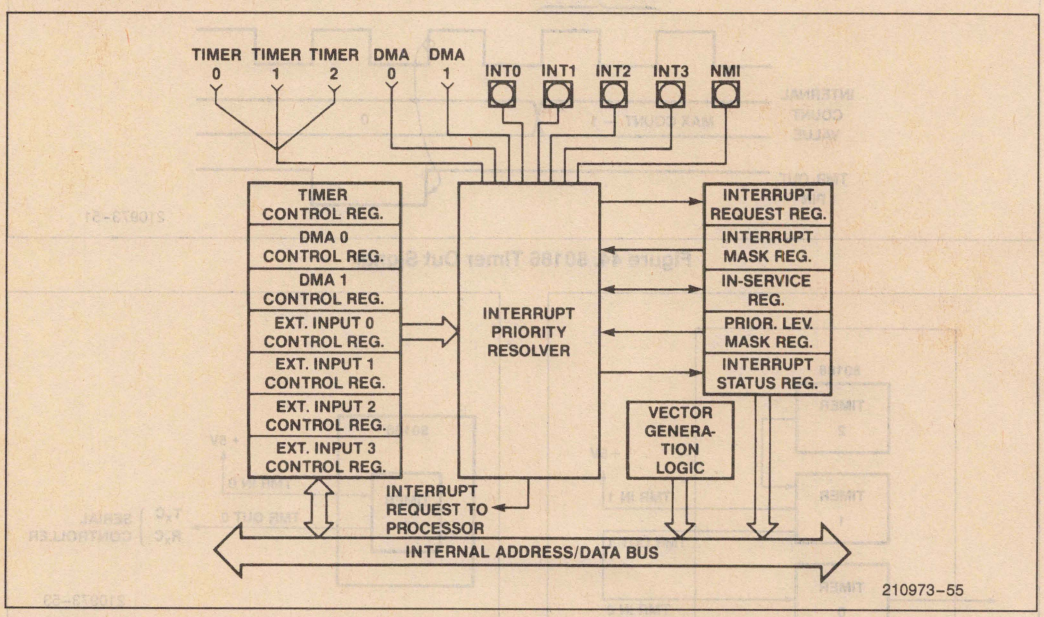


Figure 48. 80186 Interrupt Controller Block Diagram

6.0 80186 INTERRUPT CONTROLLER INTERFACING

The 80186 contains an integrated interrupt controller. This unit performs tasks of the interrupt controller in a typical system. These include synchronization of interrupt requests, prioritization of interrupt requests, and request type vectoring in response to a CPU interrupt acknowledge. It can be a master to two external 8259A interrupt controllers or can be a slave to an external interrupt controller to allow compatibility with the iRMX 86 operating system, and the 80130/80150 operating system firmware chips.

6.1 Interrupt Controller Model

The integrated interrupt controller block diagram is shown in Figure 48. It contains registers and a control element. Four inputs are provided for external interfacing to the interrupt controller. Their functions change according to the programmed mode of the interrupt controller. Like the other 80186 integrated peripheral registers, the interrupt controller registers are available for CPU reading or writing at any time.

6.2 Interrupt Controller Operation

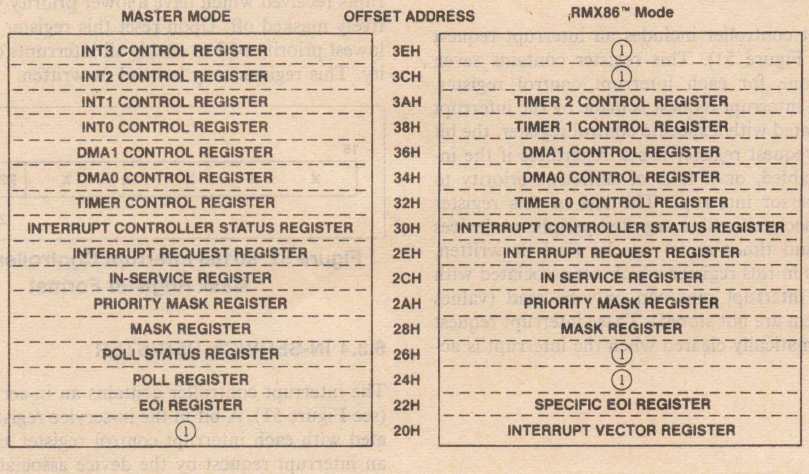
The interrupt controller operates in two major modes, non-iRMX 86™ mode (referred to henceforth as **master** mode), and **iRMX 86** mode. In master mode the integrated controller acts as the master interrupt controller for the system, while in iRMX 86 mode the controller operates as a slave to an external interrupt controller which operates as the master interrupt controller for the system. Some of the interrupt controller registers and interrupt controller pins change definition between these two modes, but the basic charter and function of the interrupt controller remains fundamentally the same. The difference is when in master mode, the interrupt controller presents its interrupt input directly to the 80186 CPU, while in iRMX 86 mode the interrupt controller presents its interrupt input to an external controller (which then presents its interrupt input to the 80186 CPU). Placing the interrupt controller in iRMX 86 mode is done by setting the iRMX mode bit in the peripheral control block pointer (see appendix A).

6.3 Interrupt Controller Registers

The interrupt controller has a number of registers which are used to control its operation (see Figure 49). Some of these change their function between the two major modes of the interrupt controller (master and iRMX 86 mode). The differences are indicated in the following section. If not indicated, the function and implementation of the registers is the same in the two basic modes of operation of the interrupt controller. The method of interaction among the various interrupt controller registers is shown in the flowcharts in Figures 57 and 58.

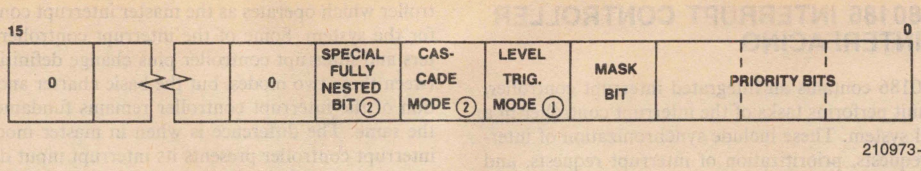
6.3.1 CONTROL REGISTERS

Each source of interrupt to the 80186 has a control register in the internal controller. These registers con-



NOTE:
1. Unsupported in this mode: values written may or may not be stored

Figure 49. 80186 Interrupt Controller Registers



NOTE:
1. This bit present only in INT0–INT3 control registers
2. These bits present only in INT0–INT1 control register

Figure 50. Interrupt Controller Control Register

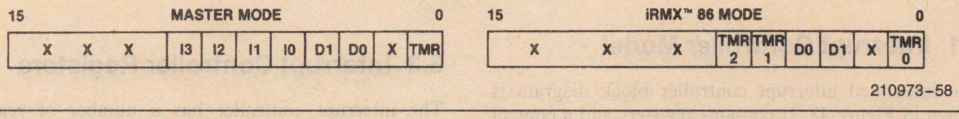


Figure 51. 80186 Interrupt Controller In-Service, Interrupt Request and Mask Register Format

tain three bits which select one of eight different interrupt priority levels for the interrupt device (0 is highest priority, 7 is lowest priority), and a mask bit to enable the interrupt (see Figure 50). When the mask bit is low, the interrupt is enabled, when it is high, the interrupt is masked.

There are seven control registers in the 80186 integrated interrupt controller. In master mode, four of these serve the external interrupt inputs, one each for the two DMA channels, and one for the collective timer interrupts. In iRMX 86 mode, the external interrupt inputs are not used, so each timer can have its own individual control register.

6.3.2 REQUEST REGISTER

The interrupt controller includes an interrupt request register (see Figure 51). This register contains seven active bits, one for each interrupt control register. Whenever an interrupt request is made by the interrupt source associated with a specific control register, the bit in interrupt request register is set, regardless if the interrupt is enabled, or if it is of sufficient priority to cause a processor interrupt. The bits in this register which are associated with integrated peripheral devices (the DMA and timer units) can be read or written, while the bits in this register which are associated with the external interrupt pins can only be read (values written to them are not stored). These interrupt request bits are automatically cleared when the interrupt is acknowledged.

6.3.3 MASK REGISTER AND PRIORITY MASK REGISTER

The interrupt controller contains a mask register (see Figure 51). This register contains a mask bit for each interrupt source associated with an interrupt control register. The bit for an interrupt source in the mask register is identically the same bit as is provided in the interrupt control register: modifying a mask bit in the control register will also modify it in the mask register, and vice versa.

The interrupt controller also contains a priority mask register (see Figure 52). This register contains three bits which indicate the lowest priority an interrupt may have that will cause an interrupt acknowledge. Interrupts received which have a lower priority will be effectively masked off. Upon reset this register is set to the lowest priority of 7 to enable all interrupts of any priority. This register may be read or written.

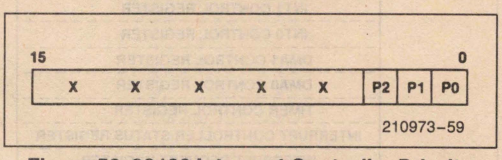


Figure 52. 80186 Interrupt Controller Priority Mask Register Format

6.3.4 IN-SERVICE REGISTER

The interrupt controller contains an in-service register (see Figure 51). A bit in the in-service register is associated with each interrupt control register so that when an interrupt request by the device associated with the

control register is acknowledged by the processor (either by the processor running the interrupt acknowledge or by the processor reading the interrupt poll register) the bit is set. The bit is reset when the CPU issues an End Of Interrupt to the interrupt controller. This register may be both read and written, i.e., the CPU may set in-service bits without an interrupt ever occurring, or may reset them without using the EOI function of the interrupt controller.

6.3.5 POLL AND POLL STATUS REGISTERS

The interrupt controller contains both a poll register and a poll status register (see Figure 53). Both of these registers contain the same information. They have a single bit to indicate an interrupt is pending. This bit is set if an interrupt of sufficient priority has been received. It is automatically cleared when the interrupt is acknowledged. If (and only if) an interrupt is pending, they also contain information as to the interrupt type of the highest priority interrupt pending.

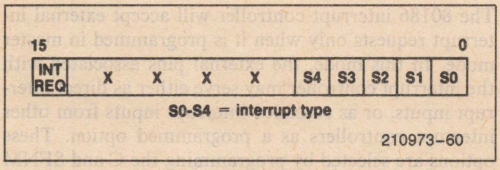


Figure 53. 80186 Poll & Poll Status Register Format

Reading the poll register will acknowledge the pending interrupt to the interrupt controller just as if the processor had acknowledged the interrupt through interrupt acknowledge cycles. The processor will not actually run any interrupt acknowledge cycles, and will not vector through a location in the interrupt vector table. Only the interrupt request, in-service and priority mask registers in the interrupt controller are set appropriately. Reading the poll status register will merely transmit the status of the polling bits without modifying any of the other interrupt controller registers. These registers are read only: data written to them is not stored. These registers are not supported in iRMX 86 mode. The state of the bits in these registers in iRMX 86 mode is not defined.

6.3.6 END OF INTERRUPT REGISTER

The interrupt controller contains an End Of Interrupt register (see Figure 54). The programmer issues an End Of Interrupt to the controller by writing to this register. After receiving the End Of Interrupt, the interrupt controller automatically resets the in-service bit for the interrupt. The value of the word written to this register determines whether the End Of Interrupt is specific or non-specific. A non-specific End Of Interrupt is specified by setting the non-specific bit in the word written to the End Of Interrupt register. In a non-specific End Of Interrupt, the in-service bit of the highest priority interrupt set is automatically cleared, while a specific End Of Interrupt allows the in-service bit cleared to be explicitly specified. The in-service bit is reset whether the bit was set by an interrupt acknowledge or if it was set by the CPU writing the bit directly to the in-service

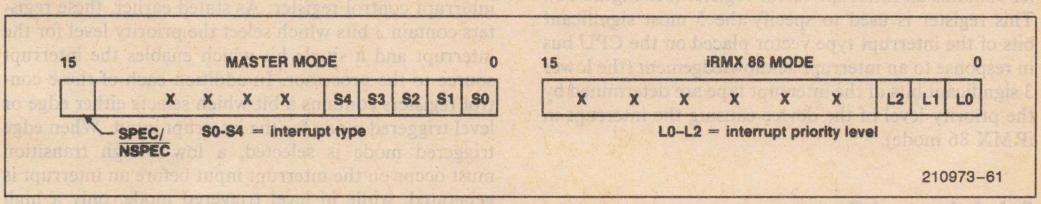


Figure 54. 80186 End of Interrupt Register Format

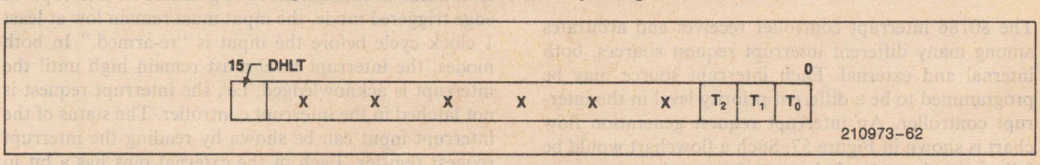


Figure 55. 80186 Interrupt Status Register Format

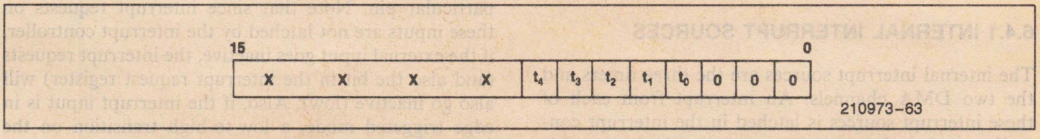


Figure 56. 80186 Interrupt Vector Register Format (iRMX 86 mode only)

register. If the highest priority interrupt is reset, the priority mask register bits will change to reflect the next lowest priority interrupt to be serviced. If a less than highest priority interrupt in-service bit is reset, the priority mask register bits will not be modified (because the highest priority interrupt being serviced has not changed). Only the specific EOI is supported in iRMX 86 mode. This register is write only: data written is not stored and cannot be read back.

6.3.7 INTERRUPT STATUS REGISTER

The interrupt controller also contains an interrupt status register (see Figure 55). This register contains four significant bits. There are three bits used to show which timer is causing an interrupt. This is required because in master mode, the timers share a single interrupt control register. A bit in this register is set to indicate which timer has generated an interrupt. The bit associated with a timer is automatically cleared after the interrupt request for the timer is acknowledged. More than one of these bits may be set at a time. The fourth bit in the interrupt status register is the DMA halt bit. When set, this bit prevents any DMA activity. It is automatically set whenever a NMI is received by the interrupt controller. It can also be set explicitly by the programmer. This bit is automatically cleared whenever the IRET instruction is executed. All significant bits in this register are read/write.

6.3.8 INTERRUPT VECTOR REGISTER

Finally, in iRMX 86 mode only, the interrupt controller contains an interrupt vector register (see Figure 56). This register is used to specify the 5 most significant bits of the interrupt type vector placed on the CPU bus in response to an interrupt acknowledgement (the lower 3 significant bits of the interrupt type are determined by the priority level of the device causing the interrupt in iRMX 86 mode).

6.4 Interrupt Sources

The 80186 interrupt controller receives and arbitrates among many different interrupt request sources, both internal and external. Each interrupt source may be programmed to be a different priority level in the interrupt controller. An interrupt request generation flow chart is shown in Figure 57. Such a flowchart would be followed independently by each interrupt source.

6.4.1 INTERNAL INTERRUPT SOURCES

The internal interrupt sources are the three timers and the two DMA channels. An interrupt from each of these interrupt sources is latched in the interrupt controller, so that if the condition causing the interrupt is cleared in the originating integrated peripheral device, the interrupt request will remain pending in the interrupt controller. The state of the pending interrupt can be obtained by reading the interrupt request register of the interrupt controller. For all internal interrupts, the latched interrupt request can be reset by the processor by writing to the interrupt request register. Note that all timers share a common bit in the interrupt request register in master mode. The interrupt controller status register may be read to determine which timer is actually causing the interrupt request in this mode. Each timer has a unique interrupt vector (see section 6.5.1). Thus polling is not required to determine which timer has caused the interrupt in the interrupt service routine. Also, because the timers share a common interrupt control register, they are placed at a common priority level as referenced to all other interrupt devices. Among themselves they have a fixed priority, with timer 0 as the highest priority timer and timer 2 as the lower priority timer.

6.4.2 EXTERNAL INTERRUPT SOURCES

The 80186 interrupt controller will accept external interrupt requests only when it is programmed in master mode. In this mode, the external pins associated with the interrupt controller may serve either as direct interrupt inputs, or as cascaded interrupt inputs from other interrupt controllers as a programmed option. These options are selected by programming the C and SFNM bits in the INT0 and INT1 control registers (see Figure 50).

When programmed as direct interrupt inputs, the four interrupt inputs are each controlled by an individual interrupt control register. As stated earlier, these registers contain 3 bits which select the priority level for the interrupt and a single bit which enables the interrupt source to the processor. In addition each of these control registers contains a bit which selects either edge or level triggered mode for the interrupt input. When edge triggered mode is selected, a low-to-high transition must occur on the interrupt input before an interrupt is generated, while in level triggered mode, only a high level needs to be maintained to generate an interrupt. In edge triggered mode, the input must remain low at least 1 clock cycle before the input is "re-armed." In both modes, the interrupt level must remain high until the interrupt is acknowledged, i.e., the interrupt request is not latched in the interrupt controller. The status of the interrupt input can be shown by reading the interrupt request register. Each of the external pins has a bit in this register which indicates an interrupt request on the particular pin. Note that since interrupt requests on these inputs are not latched by the interrupt controller, if the external input goes inactive, the interrupt requests (and also the bit in the interrupt request register) will also go inactive (low). Also, if the interrupt input is in edge triggered mode, a low-to-high transition on the input pin must occur before the interrupt request bit will be set in the interrupt request register.

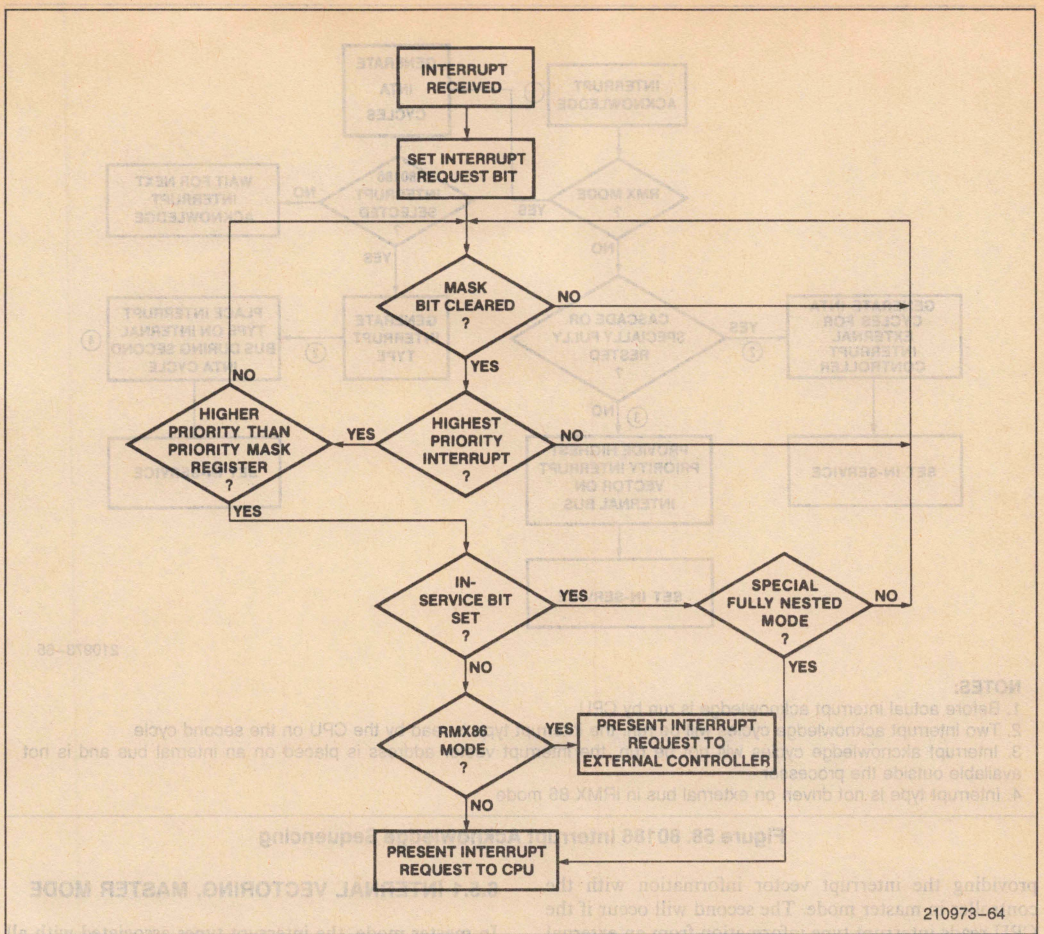


Figure 57. 80186 Interrupt Request Sequencing

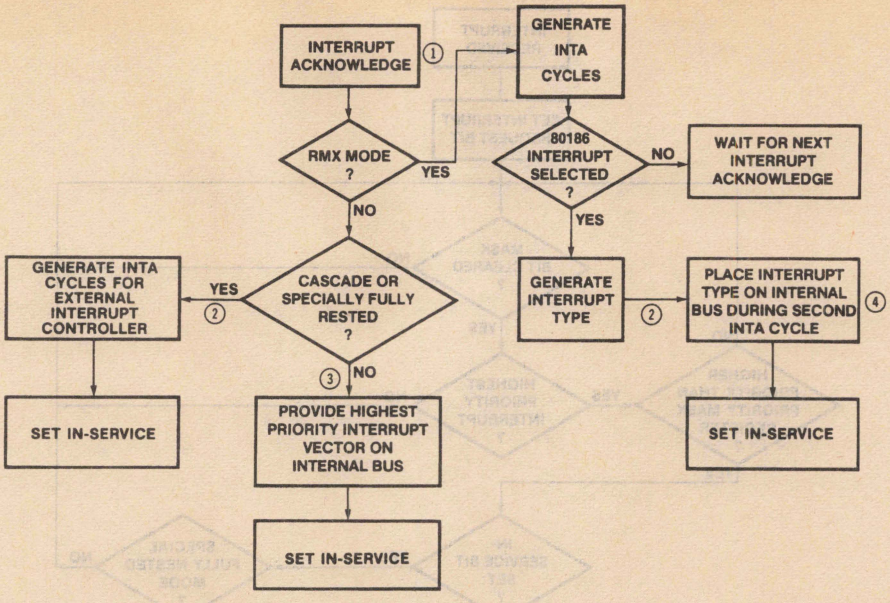
If the C (Cascade) bit of the INT0 or INT1 control registers are set, the interrupt input is cascaded to an external interrupt controller. In this mode, whenever the interrupt presented to the INT0 or INT1 line is acknowledged, the integrated interrupt controller will not provide the interrupt type for the interrupt. Instead, two INTA bus cycles will be run, with the INT2 and INT3 lines providing the interrupt acknowledge pulses for the INT0 and the INT1 interrupt requests respectively. INT0/INT2 and INT1/INT3 may be individually programmed into cascade mode. This allows 128 individually vectored interrupt sources if two banks of 9 external interrupt controllers each are used.

6.4.3 iRMX 86 MODE INTERRUPT SOURCES

When the interrupt controller is configured in iRMX 86 mode, the integrated interrupt controller accepts interrupt requests only from the integrated peripherals. Any external interrupt requests must go through an external interrupt controller. This external interrupt controller requests interrupt service directly from the 80186 CPU through the INT0 line on the 80186. In this mode, the function of this line is not affected by the integrated interrupt controller. In addition, in iRMX 86 mode the integrated interrupt controller must request interrupt service through this external interrupt controller. This interrupt request is made on the INT3 line (see section 6.7.4 on external interrupt connections).

6.5 Interrupt Response

The 80186 can respond to an interrupt in two different ways. The first will occur if the internal controller is



NOTES:
1. Before actual interrupt acknowledge is run by CPU
2. Two interrupt acknowledge cycles will be run, the interrupt type is read by the CPU on the second cycle
3. Interrupt akcnowledge cycles will not be run, the interrupt vector address is placed on an internal bus and is not available outside the processor
4. Interrupt type is not driven on external bus in iRMX 86 mode

Figure 58. 80186 Interrupt Acknowledge Sequencing

providing the interrupt vector information with the controller in master mode. The second will occur if the CPU reads interrupt type information from an external interrupt controller or if the interrupt controller is in iRMX 86 mode. In both of these instances the interrupt vector information driven by the 80186 integrated interrupt controller is not available outside the 80186 microprocessor.

In each interrupt mode, when the integrated interrupt controller receives an interrupt response, the interrupt controller will automatically set the in-service bit and reset the interrupt request bit in the integrated controller. In addition, unless the interrupt control register for the interrupt is set in Special Fully Nested Mode, the interrupt controller will prevent any interrupts from occurring from the same interrupt line until the in-service bit for that line has been cleared.

6.5.1 INTERNAL VECTORING, MASTER MODE

In master mode, the interrupt types associated with all the interrupt sources are fixed and unalterable. These interrupt types are given in Table 5. In response to an internal CPU interrupt acknowledge the interrupt controller will generate the vector address rather than the interrupt type. On the 80186 (like the 8086) the interrupt vector address is the interrupt type multiplied by 4. This speeds interrupt response.

In master mode, the integrated interrupt controller is the master interrupt controller of the system. As a result, no external interrupt controller need know when the integrated controller is providing an interrupt vector, nor when the interrupt acknowledge is taking place. As a result, no interrupt acknowledge bus cycles will be generated. The first external indication that an interrupt has been acknowledged will be the processor reading the interrupt vector from the interrupt vector table in low memory.

Table 5. 80186 Interrupt Vector Types

Interrupt Name	Vector Type	Default Priority
timer 0	8	0a
timer 1	18	0b
timer 2	19	0c
DMA 0	10	2
DMA 1	11	3
INT 0	12	4
INT 1	13	5
INT 2	14	6
INT 3	15	7

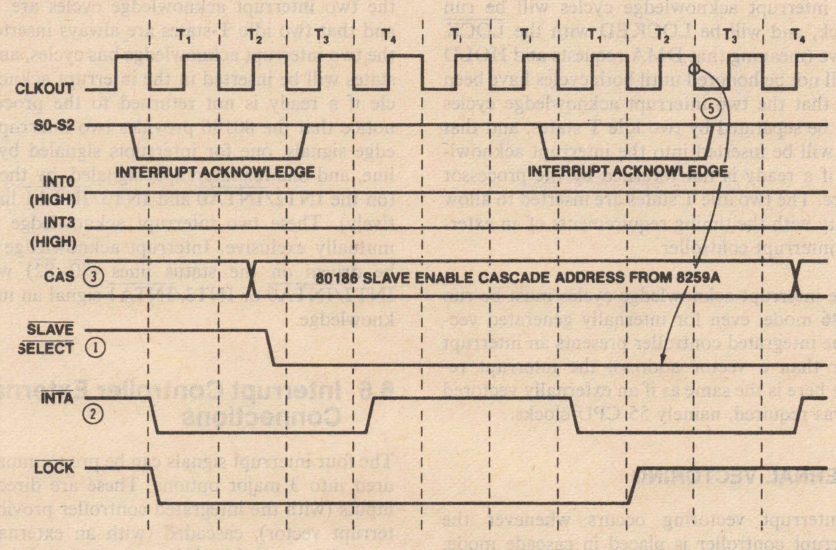
Because the two interrupt acknowledge cycles are not run, and the interrupt vector address does not need to be calculated, interrupt response to an internally vectored interrupt is 42 clock cycles, which is faster than the interrupt response when external vectoring is required, or if the interrupt controller is run in iRMX 86 mode.

If two interrupts of the same programmed priority occur, the default priority scheme (as shown in table 5) is used.

6.5.2 INTERNAL VECTORING, iRMX 86 MODE

In iRMX 86 mode, the interrupt types associated with the various interrupt sources are alterable. The upper 5 most significant bits are taken from the interrupt vector register, and the lower 3 significant bits are taken from the priority level of the device causing the interrupt. Because the interrupt type, rather than the interrupt vector address, is given by the interrupt controller in this mode the interrupt vector address must be calculated by the CPU before servicing the interrupt.

In iRMX 86 mode, the integrated interrupt controller will present the interrupt type to the CPU in response to the two interrupt acknowledge bus cycles run by the processor. During the first interrupt acknowledge cycle, the external master interrupt controller determines which slave interrupt controller will be allowed to place its interrupt vector on the microprocessor bus. During the second interrupt acknowledge cycle, the processor reads the interrupt vector from its bus. Thus, these two interrupt acknowledge cycles must be run, since the integrated controller will present the interrupt type information only when the external interrupt controller signals the integrated controller that it has the highest pending interrupt request (see Figure 59). The 80186



NOTES:
1. $\overline{\text{SLAVE SELECT}}$ = INT1
2. $\overline{\text{INTA}}$ = INT 2
3. Driven by external interrupt controller
4. $\overline{\text{SLAVE SELECT}}$ must be driven before Phase 2 of T_2 of the second INTA cycle
5. $\overline{\text{SLAVE SELECT}}$ read by 80186

Figure 59. 80186 iRMX-86 Mode Interrupt Acknowledge Timing

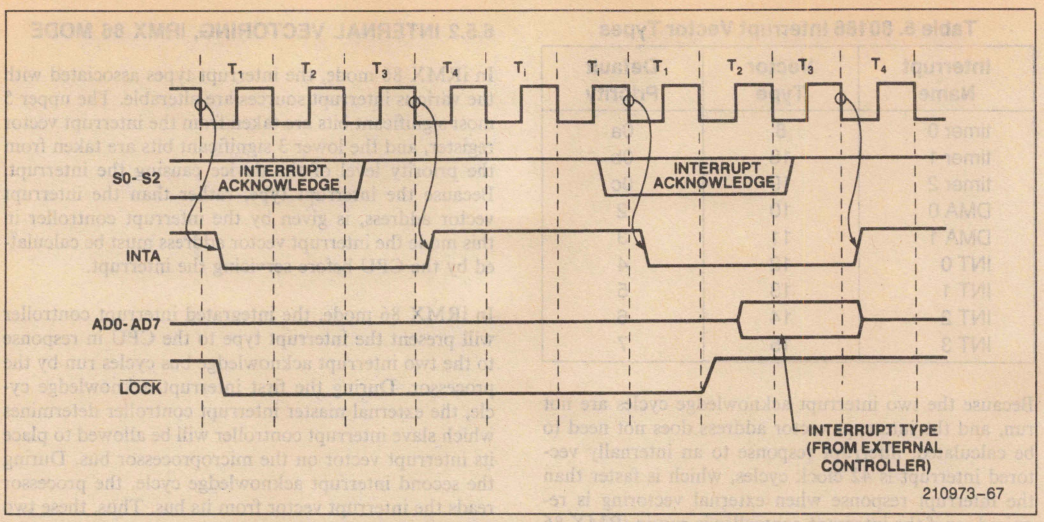


Figure 60. 80186 Cascaded Interrupt Acknowledge Timing

samples the SLAVE SELECT line during the falling edge of the clock at the beginning of T_3 of the second interrupt acknowledge cycle. This input must be stable 20 ns before and 10 ns after this edge.

These two interrupt acknowledge cycles will be run back to back, and will be LOCKED with the LOCK output active (meaning that DMA requests and HOLD requests will not be honored until both cycles have been run). Note that the two interrupt acknowledge cycles will always be separated by two idle T states, and that wait states will be inserted into the interrupt acknowledge cycle if a ready is not returned by the processor bus interface. The two idle T states are inserted to allow compatibility with the timing requirements of an external 8259A interrupt controller.

Because the interrupt acknowledge cycles must be run in iRMX 86 mode, even for internally generated vectors, and the integrated controller presents an interrupt type rather than a vector address, the interrupt response time here is the same as if an externally vectored interrupt was required, namely 55 CPU clocks.

6.5.3 EXTERNAL VECTORING

External interrupt vectoring occurs whenever the 80186 interrupt controller is placed in cascade mode, special fully nested mode, or iRMX 86 mode (and the integrated controller is not enabled by the external master interrupt controller). In this mode, the 80186 generates two interrupt acknowledge cycles, reading the interrupt type off the lower 8 bits of the address/data bus on the second interrupt acknowledge cycle (see Figure 60). This interrupt response is exactly the same as the 8086, so that the 8259A interrupt controller can be used exactly as it would in an 8086 system. Notice that the two interrupt acknowledge cycles are LOCKED, and that two idle T-states are always inserted between the two interrupt acknowledge bus cycles, and that wait states will be inserted in the interrupt acknowledge cycle if a ready is not returned to the processor. Also notice that the 80186 provides two interrupt acknowledge signals, one for interrupts signaled by the INT0 line, and one for interrupts signaled by the INT1 line (on the INT2/INTA0 and INT3/INTA1 lines, respectively). These two interrupt acknowledge signals are mutually exclusive. Interrupt acknowledge status will be driven on the status lines ($\overline{S0}$–$\overline{S2}$) when either INT2/INTA0 or INT3/INTA1 signal an interrupt acknowledge.

6.6 Interrupt Controller External Connections

The four interrupt signals can be programmably configured into 3 major options. These are direct interrupt inputs (with the integrated controller providing the interrupt vector), cascaded (with an external interrupt controller providing the interrupt vector), or iRMX 86 mode. In all these modes, any interrupt presented to the external lines must remain set until the interrupt is acknowledged.

6.6.1 DIRECT INPUT MODE

When the Cascade mode bits are cleared, the interrupt input lines are configured as direct interrupt input lines (see Figure 61). In this mode an interrupt source (e.g., an 8272 floppy disk controller) may be directly connected to the interrupt input line. Whenever an interrupt is received on the input line, the integrated controller will do nothing unless the interrupt is enabled, and it is the highest priority pending interrupt. At this time, the interrupt controller will present the interrupt to the CPU and wait for an interrupt acknowledge. When the acknowledge occurs, it will present the interrupt vector address to the CPU. In this mode, the CPU will not run any interrupt acknowledge cycles.

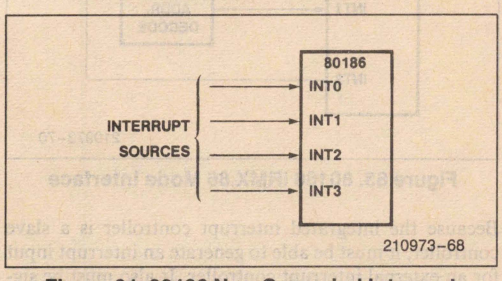


Figure 61. 80186 Non-Cascaded Interrupt Connection

These lines can be individually programmed in either edge or level triggered mode using their respective control registers. In edge triggered mode, a low-to-high transition must occur before the interrupt will be generated to the CPU, while in level triggered mode, only a high level must be present on the input for an interrupt to be generated. In edge trigger mode, the interrupt input must also be low for at least 1 CPU clock cycle to insure recognition. In both modes, the interrupt input must remain active until acknowledged.

6.6.2 CASCADE MODE

When the Cascade mode bit is set and the SFNM bit is cleared, the interrupt input lines are configured in cascade mode. In this mode, the interrupt input line is paired with an interrupt acknowledge line. The INT2/$\overline{\text{INTA0}}$ and INT3/$\overline{\text{INTA1}}$ lines are dual purpose; they can function as direct input lines, or they can function as interrupt acknowledge outputs. INT2/$\overline{\text{INTA0}}$ provides the interrupt acknowledge for an INT0 input, and INT3/$\overline{\text{INTA1}}$ provides the interrupt acknowledge for an INT1 input. Figure 62 shows this connection.

When programmed in this mode, in response to an interrupt request on the INT0 line, the 80186 will provide two interrupt acknowledge pulses. These pulses will be provided on the INT2/$\overline{\text{INTA0}}$ line, and will also be reflected by interrupt acknowledge status being generated on the $\overline{\text{S0}}$–$\overline{\text{S2}}$ status lines. On the second pulse, the interrupt type will be read in. The 80186 externally vectored interrupt response is covered in more detail in section 6.5.

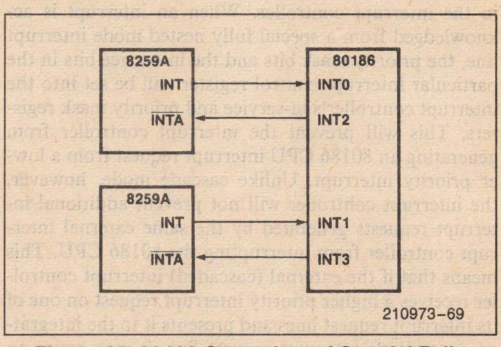


Figure 62. 80186 Cascade and Special Fully Nested Mode Interface

INT0/INT2/$\overline{\text{INTA0}}$ and INT1/INT3/$\overline{\text{INTA1}}$ may be individually programmed into interrupt request/acknowledge pairs, or programmed as direct inputs. This means that INT0/INT2/$\overline{\text{INTA0}}$ may be programmed as an interrupt/acknowledge pair, while INT1 and INT3/$\overline{\text{INTA1}}$ each provide separate internally vectored interrupt inputs.

When an interrupt is received on a cascaded interrupt, the priority mask bits and the in-service bits in the particular interrupt control register will be set into the interrupt controller's mask and priority mask registers. This will prevent the controller from generating an 80186 CPU interrupt request from a lower priority interrupt. Also, since the in-service bit is set, any subsequent interrupt requests on the particular interrupt input line will not cause the integrated interrupt controller to generate an interrupt request to the 80186 CPU. This means that if the external interrupt controller receives a higher priority interrupt request on one of its interrupt request lines and presents it to the 80186 interrupt request line, it will not subsequently be presented to the 80186 CPU by the integrated interrupt controller until the in-service bit for the interrupt line has been cleared.

6.6.3 SPECIAL FULLY NESTED MODE

When both the Cascade mode bit and the SFNM bit are set, the interrupt input lines are configured in Special Fully Nested Mode. The external interface in this mode is exactly as in Cascade Mode. The only difference is in the conditions allowing an interrupt from the external interrupt controller to the integrated interrupt controller to interrupt the 80186 CPU.

When an interrupt is received from a special fully nested mode interrupt line, it will interrupt the 80186 CPU if it is the highest priority interrupt pending regardless of the state of the in-service bit for the interrupt source in the interrupt controller. When an interrupt is acknowledged from a special fully nested mode interrupt line, the priority mask bits and the in-service bits in the particular interrupt control register will be set into the interrupt controller's in-service and priority mask registers. This will prevent the interrupt controller from generating an 80186 CPU interrupt request from a lower priority interrupt. Unlike cascade mode, however, the interrupt controller will not prevent additional interrupt requests generated by the same external interrupt controller from interrupting the 80186 CPU. This means that if the external (cascaded) interrupt controller receives a higher priority interrupt request on one of its interrupt request lines and presents it to the integrated controller's interrupt request line, it may cause an interrupt to be generated to the 80186 CPU, regardless of the state of the in-service bit for the interrupt line.

If the SFNM mode bit is set and the Cascade mode bit is not also set, the controller will provide internal interrupt vectoring. It will also ignore the state of the in-service bit in determining whether to present an interrupt request to the CPU. In other words, it will use the SFNM conditions of interrupt generation with an internally vectored interrupt response, i.e., if the interrupt pending is the highest priority type pending, it will cause a CPU interrupt regardless of the state of the in-service bit for the interrupt.

6.6.4 iRMX 86 MODE

When the RMX bit in the peripheral relocation register is set, the interrupt controller is set into iRMX 86 mode.

In this mode, all four interrupt controller input lines are used to perform the necessary handshaking with the external master interrupt controller. Figure 63 shows the hardware configuration of the 80186 interrupt lines with an external controller in iRMX 86 mode.

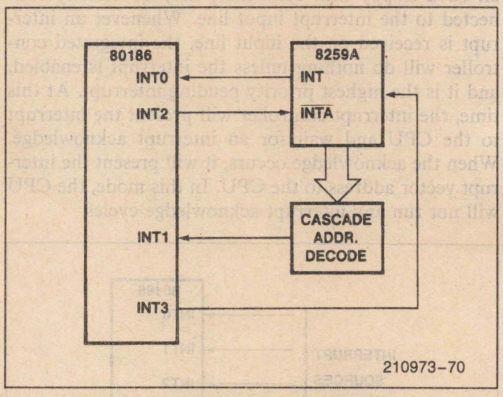


Figure 63. 80186 iRMX 86 Mode Interface

Because the integrated interrupt controller is a slave controller, it must be able to generate an interrupt input for an external interrupt controller. It also must be signaled when it has the highest priority pending interrupt to know when to place its interrupt vector on the bus. These two signals are provided by the INT3/Slave Interrupt Output and INT1/Slave Select lines, respectively. The external master interrupt controller must be able to interrupt the 80186 CPU, and needs to know when the interrupt request is acknowledged. The INT0 and INT2/INTA0 lines provide these two functions.

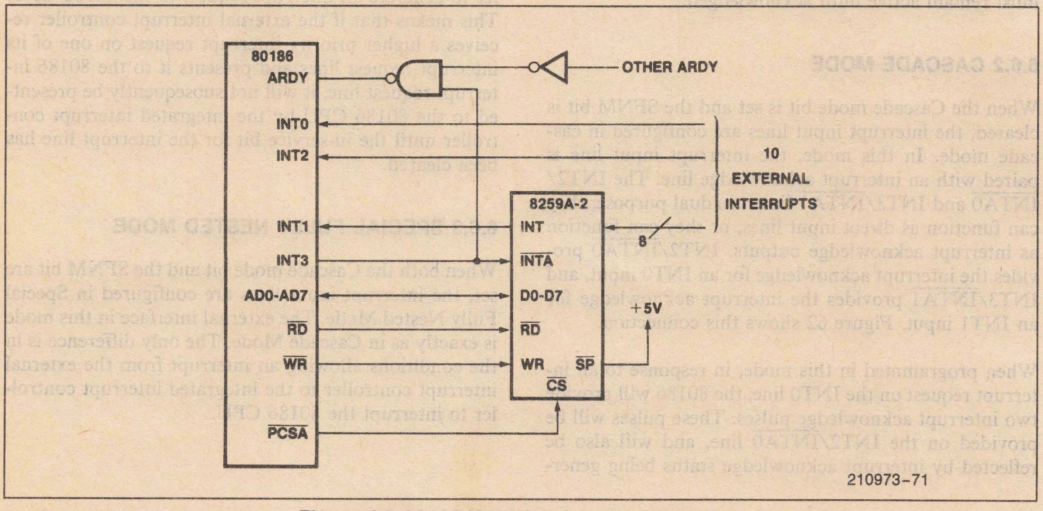


Figure 64. 80186/8259A Interrupt Cascading

6.7 Example 8259A/Cascade Mode Interface

Figure 64 shows the 80186 and 8259A in cascade interrupt mode. The code to initialize the 80186 interrupt controller is given in Appendix E. Notice that an "interrupt ready" signal must be returned to the 80186 to prevent the generation of wait states in response to the interrupt acknowledge cycles. In this configuration the INT0 and INT2 lines are used as direct interrupt input lines. Thus, this configuration provides 10 external interrupt lines: 2 provided by the 80186 interrupt controller itself, and 8 from the external 8259A. Also, the 8259A is configured as a master interrupt controller. It will only receive interrupt acknowledge pulses in response to an interrupt it has generated. It may be cascaded again to up to 8 additional 8259As (each of which would be configured in slave mode).

6.8 Example 80130 iRMX 86 Mode Interface

Figure 65 shows the 80186 and 80130 connected in iRMX 86 mode. In this mode, the 80130 interrupt controller is the master interrupt controller of the system. The 80186 generates an interrupt request to the 80130 interrupt controller when one of the 80186 integrated peripherals has created an interrupt condition, and that condition is sufficient to generate an interrupt from the 80186 integrated interrupt controller. Note that the 80130 decodes the interrupt acknowledge status directly from the 80186 status lines; thus, the INT2/INTA0 line of the 80186 need not be connected to the 80130. Figure 65 uses this interrupt acknowledge signal to enable the cascade address decoder. The 80130 drives the cascade address on the AD8–AD10 during T_1 of the second interrupt acknowledge cycle. This cascade address is latched into the system address latches, and if the proper cascade address is decoded by the 8205 decoder, the 80186 INT1/SLAVE SELECT line will be driven active, enabling the 80186 integrated interrupt controller to place its interrupt vector on the internal bus. The code to configure the 80186 into iRMX 86 mode is presented in appendix E.

6.9 Interrupt Latency

Interrupt latency time is the time from when the 80186 receives the interrupt to the time it begins to respond to the interrupt. This is different from interrupt response.

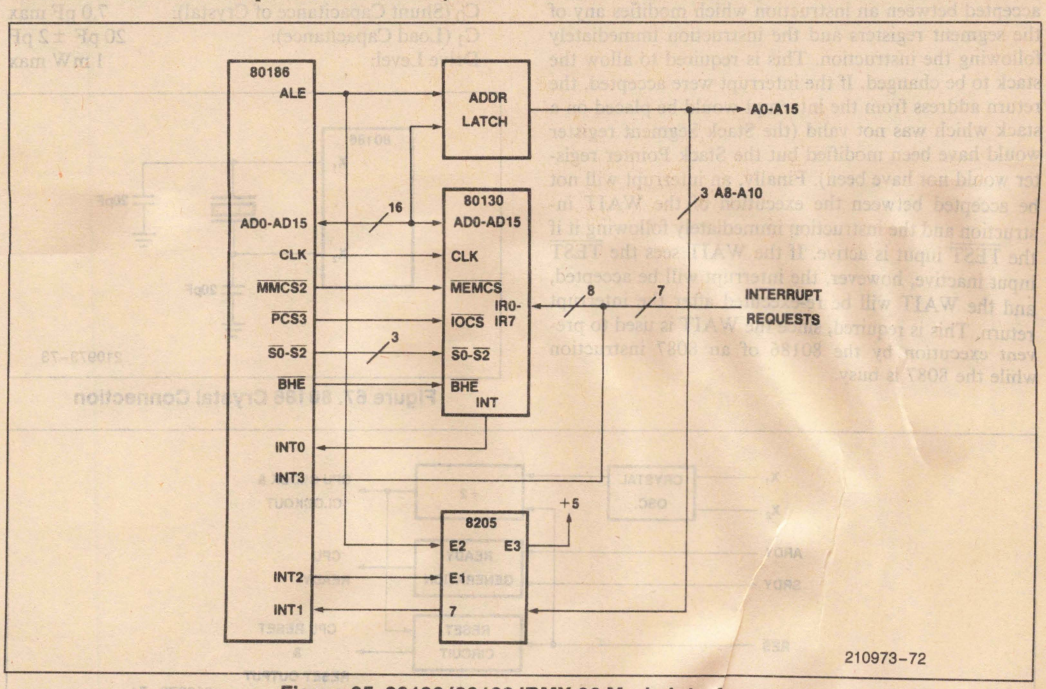


Figure 65. 80186/80130 iRMX 86 Mode Interface

time, which is the time from when the processor actually begins processing the interrupt to when it actually executes the first instruction of the interrupt service routine. The factors affecting interrupt latency are the intstruction being executed and the state of the interrupt enable flip-flop.

Interrupts will be acknowledged only if the interrupt enable flip-flop in the CPU is set. Thus, interrupt latency will be very long indeed if interrupts are never enabled by the processor!

When interrupts are enabled in the CPU, the interrupt latency is a function of the instructions being executed. Only repeated instructions will be interrupted before being completed, and those only between their respective iterations. This means that the interrupt latency time could be as long as 69 CPU clocks, which is the time it takes the processor to execute an integer divide instruction (with a segment override prefix, see below), the longest single instruction on the 80186.

Other factors can affect interrupt latency. An interrupt will not be accepted between the execution of a prefix (such as segment override prefixes and lock prefixes) and the instruction. In addition, an interrupt will not be accepted between an instruction which modifies any of the segment registers and the instruction immediately following the instruction. This is required to allow the stack to be changed. If the interrupt were accepted, the return address from the interrupt would be placed on a stack which was not valid (the Stack Segment register would have been modified but the Stack Pointer register would not have been). Finally, an interrupt will not be accepted between the execution of the WAIT instruction and the instruction immediately following it if the $\overline{\text{TEST}}$ input is active. If the WAIT sees the $\overline{\text{TEST}}$ input inactive, however, the interrupt will be accepted, and the WAIT will be re-executed after the interrupt return. This is required, since the WAIT is used to prevent execution by the 80186 of an 8087 instruction while the 8087 is busy.

7.0 CLOCK GENERATOR

The 80186 includes a clock generator which generates the main clock signal for all 80186 integrated components, and all CPU synchronous devices in the 80186 system. This clock generator includes a crystal oscillator, divide by two counter, reset circuitry, and ready generation logic. A block diagram of the clock generator is shown in Figure 66.

7.1 Crystal Oscillator

The 80186 crystal oscillator is a parallel resonant, Pierce oscillator. It was designed to be used as shown in Figure 67. The capacitor values shown are approximate. As the crystal frequency drops, they should be increased, so that at the 4 MHz minimum crystal frequency supported by the 80186 they take on a value of 30 pF. The output of this oscillator is not directly available outside the 80186.

The following parameters may be used for choosing a crystal:

Temperature Range:	0 to 70°C
ESR (Equivalent Series Resistance):	30Ω max
C_0 (Shunt Capacitance of Crystal):	7.0 pF max
C_1 (Load Capacitance):	20 pF ±2 pF
Drive Level:	1 mW max

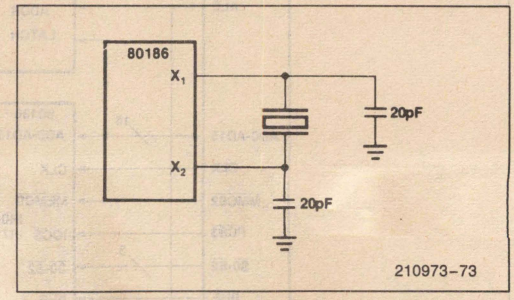
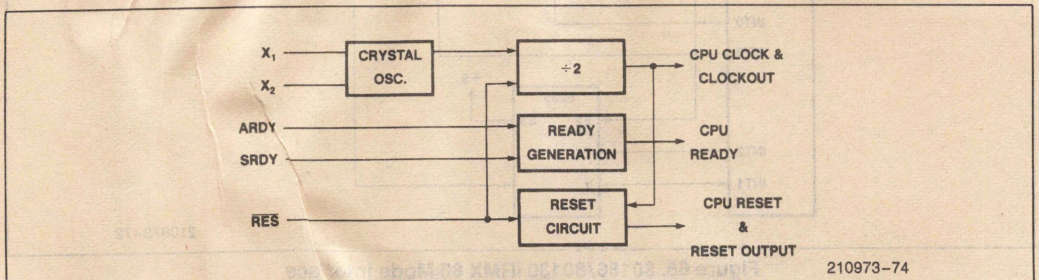


Figure 67. 80186 Crystal Connection



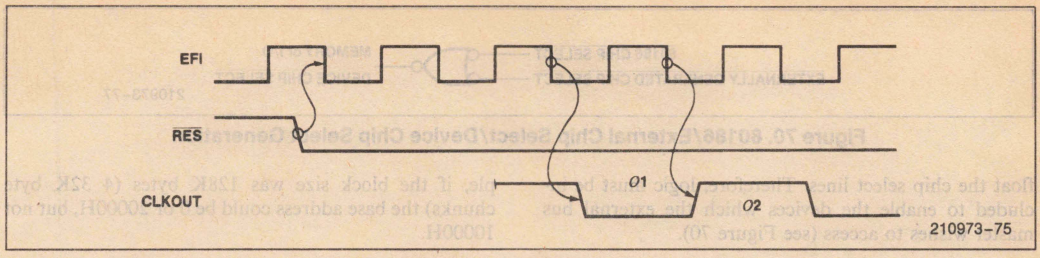


Figure 68. 80186 Clock Generator Reset

7.2 Using an External Oscillator

An external oscillator may be used with the 80186. The external frequency input (EFI) signal is connected directly to the X1 input of the oscillator. X2 should be left open. This oscillator input is used to drive an internal divide-by-two counter to generate the CPU clock signal, so the external frequency input can be of practically any duty cycle, so long as the minimum high and low times for the signal (as stated in the data sheet) are met.

7.3 Clock Generator

The output of the crystal oscillator (or the external frequency input) drives a divide by two circuit which generates a 50% duty cycle clock for the 80186 system. All 80186 timing is referenced to this signal, which is available on the CLKOUT pin of the 80186. This signal will change state on the high-to-low transition of the EFI signal.

7.4 Ready Generation

The clock generator also includes the circuitry required for ready generation. Interfacing to the SRDY and ARDY inputs this provides is covered in section 3.1.6.

7.5 Reset

The 80186 clock generator also provides a synchronized reset signal for the system. This signal is generated from the reset input ($\overline{RES}$) to the 80186. The clock generator synchronizes this signal to the clockout signal.

The reset input signal also resets the divide-by-two counter. A one clock cycle internal clear pulse is generated when the $\overline{RES}$ input signal first goes active. This clear pulse goes active beginning on the first low-to-high transition of the X1 input after $\overline{RES}$ goes active, and goes inactive on the next low-to-high transition of the X1 input. In order to insure that the clear pulse is generated on the next EFI cycle, the $\overline{RES}$ input signal must satisfy a 25 ns setup time to the high-to-low EFI input signal (see Figure 68). During this clear, clockout will be high. On the next high-to-low transition of X1, clockout will go low, and will change state on every subsequent high-to-low transition of EFI.

The reset signal presented to the rest of the 80186, and also the signal present on the RESET output pin of the 80186 is synchronized by the high-to-low transition of the clockout signal of the 80186. This signal remains active as long as the $\overline{RES}$ input also remains active. After the $\overline{RES}$ input goes inactive, the 80186 will begin to fetch its first instruction (at memory location FFFF0H) after 6 1/2 CPU clock cycles (i.e., T_1 of the first instruction fetch will occur 6 1/2 clock cycles later). To insure that the RESET ouput will go inactive on the next CPU clock cycle, the inactive going edge of the $\overline{RES}$ input must satisfy certain hold and setup times to the low-to-high edge of the clockout signal of the 80186 (see Figure 69).

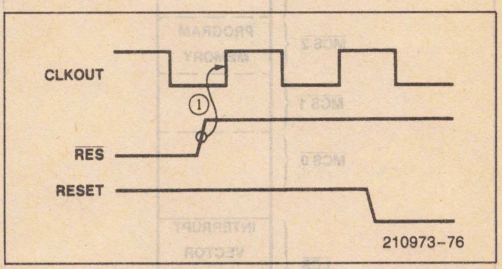


Figure 69. 80186 Coming out of Reset

8.0 CHIP SELECTS

The 80186 includes a chip select unit which generates hardware chip select signals for memory and I/O accesses generated by the 80186 CPU and DMA units. This unit is programmable such that it can be used to fulfill the chip select requirements (in terms of memory device or bank size and speed) of most small and medium sized 80186 systems.

The chip selects are driven only for internally generated bus cycles. Any cycles generated by an external unit (e.g., an external DMA controller) will not cause the chip selects to go active. Thus, any external bus masters must be responsible for their own chip select generation. Also, during a bus HOLD, the 80186 does not

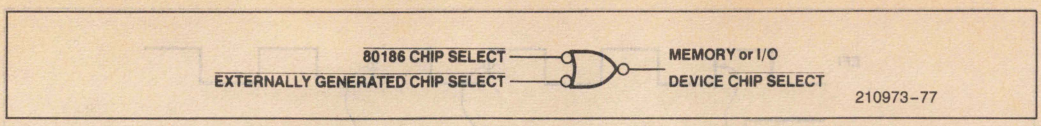


Figure 70. 80186/External Chip Select/Device Chip Select Generation

float the chip select lines. Therefore, logic must be included to enable the devices which the external bus master wishes to access (see Figure 70).

8.1 Memory Chip Selects

The 80186 provides six discrete chip select lines which are meant to be connected to memory components in an 80186 system. These signals are named $\overline{UCS}$, $\overline{LCS}$, and $\overline{MCS0\text{-}3}$ for Upper Memory Chip Select, Lower Memory Chip Select and Midrange Memory Chip Select 0–3. They are meant (but not limited) to be connected to the three major areas of the 80186 system memory (see Figure 71).

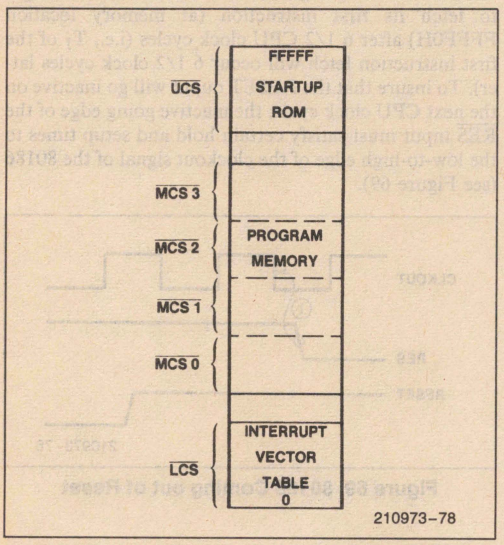


Figure 71. 80186 Memory Areas & Chip Selects

As could be guessed by their names, upper memory, lower memory, and mid-range memory chip selects are designed to address upper, lower, and middle areas of memory in an 80186 system. The upper limit of $\overline{UCS}$ and the lower limit of $\overline{LCS}$ are fixed at FFFFFH and 00000H in memory space, respectively. The other limit of these is set by the memory size programmed into the control register for the chip select line. Mid-range memory allows both the base address and the block size of the memory area to be programmed. The only limitation is that the base address must be programmed to be an integer multiple of the total block size. For example, if the block size was 128K bytes (4 32K byte chunks) the base address could be 0 or 20000H, but not 10000H.

The memory chip selects are controlled by 4 registers in the peripheral control block (see Figure 72). These include 1 each for $\overline{UCS}$ and $\overline{LCS}$, the values of which determine the size of the memory blocks addressed by these two lines. The other two registers are used to control the size and base address of the mid-range memory block.

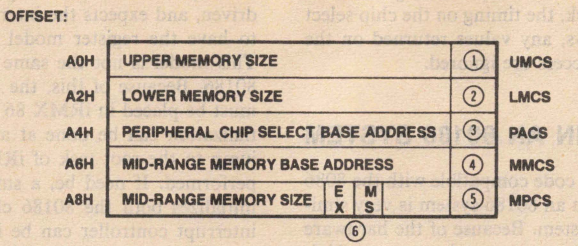
On reset, only $\overline{UCS}$ is active. It is programmed by reset to be active for the top 1K memory block, to insert 3 wait states to all memory fetches, and to factor external ready for every memory fetch (see section 8.3 for more information on internal ready generation). All other chip select registers assume indeterminate states after reset, but none of the other chip select lines will be active until all necessary registers for a signal have been accessed (not necessarily written, a read to an uninitialized register will enable the chip select function controlled by that register).

8.2 Peripheral Chip Selects

The 80186 provides seven discrete chip select lines which are meant to be connected to peripheral components in an 80186 system. These signals are named $\overline{PCS0\text{–}6}$. Each of these lines is active for one of seven continuous 128 byte areas in memory or I/O space above a programmed base address.

The peripheral chip selects are controlled by two registers in the internal peripheral control block (see Figure 72). These registers allow the base address of the peripherals to be set, and allow the peripherals to be mapped into memory or I/O space. Both of these registers must be accessed before any of the peripheral chip selects will become active.

A bit in the MPCS register allows $\overline{PCS5}$ and $\overline{PCS6}$ to become latched A1 and A2 outputs. When this option is selected, $\overline{PCS5}$ and $\overline{PCS6}$ will reflect the state of A1 and A2 throughout a bus cycle. These are provided to allow external peripheral register selection in a system in which the addresses are not latched. Upon reset, these lines are driven high. They will only reflect A1 and A2 after both PACS and MPCS have been accesssed (and are programmed to provide A1 and A2!).



NOTES:
1. Upper memory ready bits
2. Lower memory ready bits
3. PCS0–PCS3 ready bits
4. Mid-range memory ready bits
5. PCS4–PCS6 ready bits
6. MS: 1 = Peripherals active in memory space
   0 = Peripherals active in I/O space
   EX:1 = 7 PCS lines
   0 = PCS5 = A1, PCS6 = A2

Not all bits of every field are used

Figure 72. 80186 Chip Select Control Registers

8.3 Ready Generation

The 80186 includes a ready generation unit. This unit generates an internal ready signal for all accesses to memory or I/O areas to which the chip select circuitry of the 80186 responds.

For each ready generation area, 0–3 wait states may be inserted by the internal unit. Table 6 shows how the ready control bits should be programmed to provide this. In addition, the ready generation circuit may be programmed to ignore the state of the external ready (i.e., only the internal ready circuit will be used) or to factor the state of the external ready (i.e., a ready will be returned to the processor only after both the internal ready circuit has gone ready and the external ready has gone ready). Some kind of circuit must be included to generate an external ready, however, since upon reset the ready generator is programmed to factor external ready to all accesses to the top 1k byte memory block. If a ready was not returned on one of the external ready lines (ARDY or SRDY) the processor would wait forever to fetch its first instruction.

Table 6. 80186 Wait State Programming

R2	R1	R0	Number of Wait States
0	0	0	0 + external ready
0	0	1	1 + external ready
0	1	0	2 + external ready
0	1	1	3 + external ready
1	0	0	0 (no external ready required)
1	0	1	1 (no external ready required)
1	1	0	2 (no external ready required)
1	1	1	3 (no external ready required)

8.4 Examples of Chip Select Usage

Many examples of the use of the chip select lines are given in the bus interface section of this note (section 3.2). These examples show how simple it is to use the chip select function provided by the 80186. The key point to remember when using the chip select function is that they are only activated during bus cycles generated by the 80186 CPU or DMA units. When another master has the bus, it must generate its own chip select function. In addition, whenever the bus is given by the 80186 to an external master (through the HOLD/HLDA arrangement) the 80186 does NOT float the chip select lines.

8.5 Overlapping Chip Select Areas

Generally, the chip selects of the 80186 should not be programmed such that any two areas overlap. In addition, none of the programmed chip select areas should overlap any of the locations of the integrated 256-byte control register block. The consequences of doing this are:

Whenever two chip select lines are programmed to respond to the same area, both will be activated during any access to that area. When this is done, the ready bits for both areas *must* be programmed to the same value. If this is not done, the processor response to an access in this area is indeterminate. This rule also applies to overlapping chip selects with the integrated control block.

If any of the chip select area overlap the integrated 256-byte control register block, the timing on the chip select line is altered. As always, any values returned on the external bus from this access are ignored.

9.0 SOFTWARE IN AN 80186 SYSTEM

Since the 80186 is object code compatible with the 8086 and 8088, the software in an 80186 system is very similar to that in an 8086 system. Because of the hardware chip select functions, however, a certain amount of initialization code must be included when using those functions on the 80186.

9.1 System Initialization in an 80186 System

Most programmable components of a computer system must be initialized before they are used. This is also true for the 80186. The 80186 includes circuitry which directly affects the ability of the system to address memory and I/O devices, namely the chip select circuitry. This circuitry must be initialized before the memory areas and peripheral devices addressed by the chip select signals are used.

Upon reset, the UMCS register is programmed to be active for all memory fetches within the top 1K byte of memory space. It is also programmed to insert three wait states to all memory accesses within this space. If the hardware chip selects are used, they must be programmed before the processor leaves this 1K byte area of memory. If a jump to an area for which the chips are not selected occurs, the microcomputer system will cease to operate (since the processor will fetch garbage from the data bus). Appendix F shows a typical initialization sequence for the 80186 chip select unit.

Once the chip selects have been properly initialized, the rest of the 80186 system may be initialized much like an 8086 system. For example, the interrupt vector table might get set up, the interrupt controller initialized, a serial I/O channel initialized, and the main program begun. Note that the integrated peripherals included in the 80186 do not share the same programming model as the standard Intel peripherals used to implement these functions in a typical 8086 system, i.e. different values must be programmed into different registers to achieve the same function using the integrated peripherals. Appendix F shows a typical initialization sequence for an interrupt driven system using the 80186 interrupt controller.

9.2 Initialization for iRMX 86 System

Using the iRMX 86 operating system with the 80186 requires an external 8259A and an external 8253/4 or alternatively an external 80130 OSF component. These are required because the operating system is interrupt driven, and expects the interrupt controller and timers to have the register model of these external devices. This model is not the same as is implemented by the 80186. Because of this, the 80186 interrupt controller must be placed in iRMX 86 mode after reset. This initialization can be done at any time after reset before jump to the root task of iRMX 86 System is actually performed. If need be, a small section of code which initializes both the 80186 chip selects and the 80186 interrupt controller can be inserted between the reset vector location and the beginning of iRMX 86 System (see Figure 73). In this case, upon reset, the processor would jump to the 80186 initialization code, and when this has been completed, would jump to the iRMX 86 initialization code (in the root task). It is important that the 80186 hardware be initialized before iRMX 86 operation is begun, since some of the resources addressed by the 80186 system may not be initialized properly by iRMX 86 System if the initialization is done in the reverse manner.

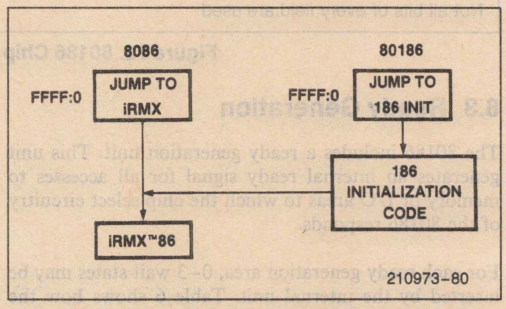


Figure 73. iRMX-86 Initialization with 8086 and 80186

9.3 Instruction Execution Differences between the 8086 and 80186

There are a few instruction execution differences between the 8086 and the 80186. These differences are:

UNDEFINED OPCODES:

When the opcodes 63H, 64H, 65H, 66H, 67H, F1H, FEH XX111XXXB and FFH XX111XXXB are executed, the 80186 will execute an illegal instruction exception, interrupt type 6. The 8086 will ignore the opcode.

0FH OPCODE:

When the opcode 0FH is encountered, the 8086 will execute a POP CS, while the 80186 will excecute an illegal instruction exception, interrupt type 6.

WORD WRITE AT OFFSET FFFFH:

When a word write is performed at offset FFFFH in a segment, the 8086 will write one byte at offset FFFFH, and the other at offset 0, while the 80186 will write one byte at offset FFFFH, and the other at offset 10000H (one byte beyond the end of the segment). One byte segment underflow will also occur (on the 80186) if a stack PUSH is executed and the Stack Pointer contains the value 1.

SHIFT/ROTATE BY VALUE GREATER THAN 31:

Before the 80186 performs a shift or rotate by a value (either in the CL register, or by an immediate value) it ANDs the value with 1FH, limiting the number of bits rotated to less than 32. The 8086 does not do this.

LOCK PREFIX:

The 8086 activates its LOCK signal immediately after executing the LOCK prefix. The 80186 does not activate the LOCK signal until the processor is ready to begin the data cycles associated with the LOCKed instruction.

NOTE:
When executing more than one LOCKed instruction, always make sure there are 6 bytes of code between the end of the first LOCKed instruction and the start of the second LOCKed instruction.

INTERRUPTED STRING MOVE INSTRUCTIONS:

If an 8086 is interrupted during the execution of a repeated string move instruction, the return value it will push on the stack will point to the last prefix instruction before the string move instruction. If the instruction had more than one prefix (e.g., a segment override prefix in addition to the repeat prefix), it will not be re-executed upon returning from the interrupt. The 80186 will push the value of the first prefix to the repeated instruction, so long as prefixes are not repeated, allowing the string instruction to properly resume.

CONDITIONS CAUSING DIVIDE ERROR WITH AN INTEGER DIVIDE:

The 8086 will cause a divide error whenever the absolute value of the quotient is greater than 7FFFH (for word operations) or if the absolute value of the quotient is greater than 7FH (for byte operations). The 80186 has expanded the range of negative numbers allowed as a quotient by 1 to include 8000H and 80H. These numbers represent the most negative numbers representable using 2's complement arithmetic (equaling −32768 and −128 in decimal, respectively).

ESC OPCODE:

The 80186 may be programmed to cause an interrupt type 7 whenever an ESCape instruction (used for coprocessors like the 8087) is executed. The 8086 has no such provision. Before the 80186 performs this trap, it must be programmed to do so.

These differences can be used to determine whether the program is being executed on an 8086 or an 80186. Probably the safest execution difference to use for this purpose is the difference in multiple bit shifts. For example, if a multiple bit shift is programmed where the shift count (stored in the CL register!) is 33, the 8086 will shift the value 33 bits, whereas the 80186 will shift it only a single bit.

In addition to the instruction execution differences noted above, the 80186 includes a number of new instruction types, which simplify assembly language programming of the processor, and enhance the performance of higher level languages running on the processor. These new instructions are covered in depth in the 8086/80186 users manual and in appendix H of this note.

10.0 CONCLUSIONS

The 80186 is a glittering example of state-of-the-art integrated circuit technology applied to make the job of the microprocessor system designer simpler and faster. Because many of the required peripherals and their interfaces have been cast in silicon, and because of the timing and drive latitudes provided by the part, the designer is free to concentrate on other issues of system design. As a result, systems designed around the 80186 allow applications where no other processor has been able to provide the necessary performance at a comparable size or cost.

APPENDIX A
PERIPHERAL CONTROL BLOCK

All the integrated peripherals within the 80186 microprocessor are controlled by sets of registers contained within an integrated peripheral control block. The registers are physically located within the peripheral devices they control, but are addressed as a single block of registers. This set of registers fills 256 contiguous bytes and can be located beginning on any 256 byte boundary of the 80186 memory or I/O space. A map of these registers is shown in Figure A-1.

A.1 SETTING THE BASE LOCATION OF THE PERIPHERAL CONTROL BLOCK

In addition to the control registers for each of the integrated 80186 peripheral devices, the peripheral control block contains the peripheral control block relocation register. This register allows the peripheral control block to be re-located on any 256 byte boundary within the processor's memory or I/O space. Figure A-2 shows the layout of this register.

This register is located at offset FEH within the peripheral control block. Since it is itself contained within the peripheral control block, any time the location of the peripheral control block is moved, the location of the relocation registers will also move.

In addition to the peripheral control block relocation information, the relocation register contains two additional bits. One is used to set the interrupt controller into iRMX 86 compatibility mode. The other is used to force the processor to trap whenever an ESCape (coprocessor) instruction is encountered.

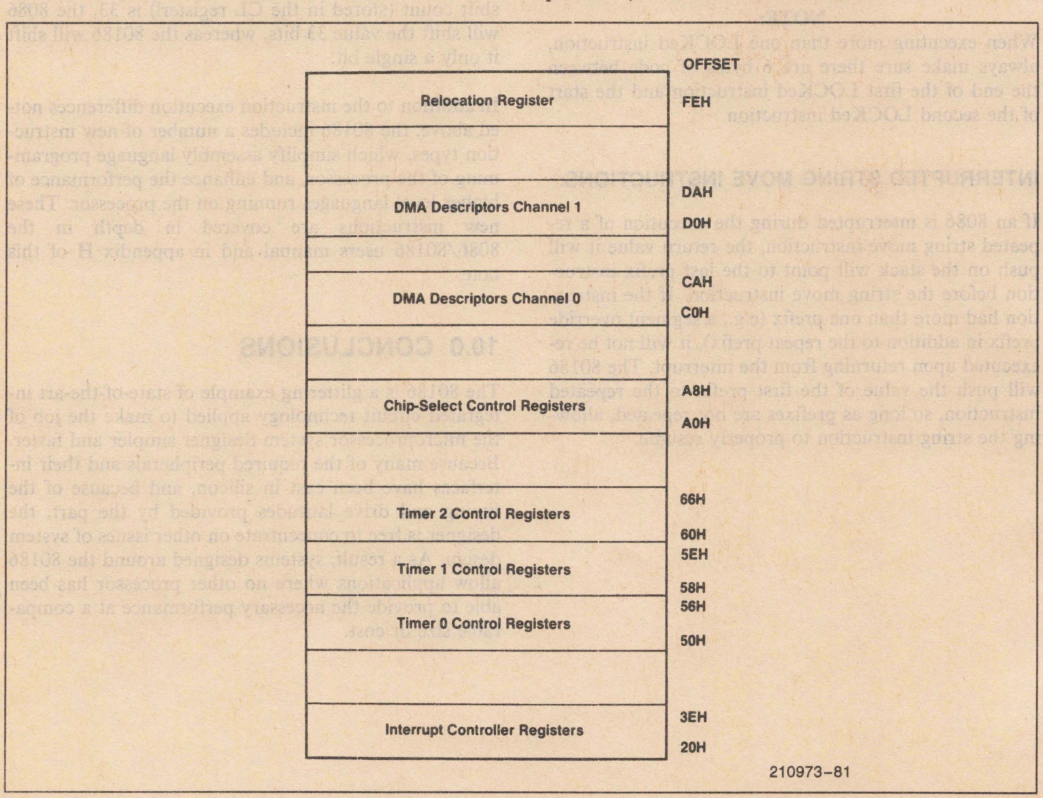
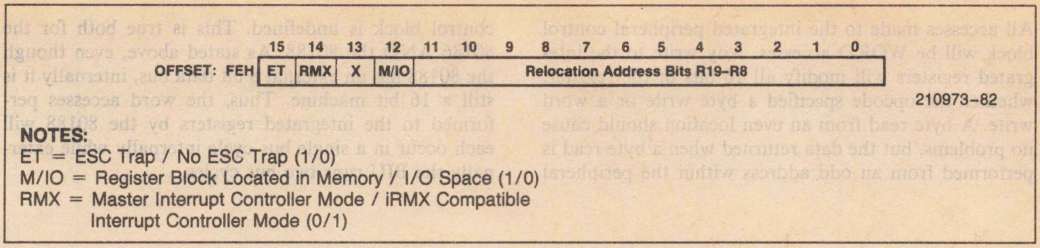


Figure A-1. 80186 Integrated Peripheral Control Block



NOTES:
ET = ESC Trap / No ESC Trap (1/0)
M/IO = Register Block Located in Memory / I/O Space (1/0)
RMX = Master Interrupt Controller Mode / iRMX Compatible Interrupt Controller Mode (0/1)

Figure A-2. 80186 Relocation Register Layout

Because the relocation register is contained within the peripheral control block, upon reset the relocation register is automatically programmed with the value 20 FFH. This means that the peripheral control block will be located at the very top (FF00H to FFFFH) of I/O space. Thus, after reset the relocation register will be located at word location FFFEH in I/O space.

If the user wished to locate the peripheral control block starting at memory location 10000H he would program the peripheral control register with the value 1100H. By doing this, he would move all registers within the integrated peripheral control block to memory locations 10000H to 100FFH. Note that since the relocation register is contained within the peripheral control block, it too would move to word location 100FEH in memory space.

Whenever mapping the 188 peripheral control block to another location, the programming of the relocation register should be done with a byte write (i.e., OUT DX,AL). Any access to the control block is done 16 bits at a time. Thus, internally, the relocation register will get written with 16 bits of the AX register while externally, the BIU will run only one 8 bit bus cycle. If a word instruction is used (i.e., OUT DX,AX), the relocation register will be written on the first bus cycle. The BIU will then run a second bus cycle which is unnecessary. The address of the second bus cycle will no longer be within the control block (i.e., the control block was moved on the first cycle), and therefore, will require the generation of an external ready signal to complete the cycle. For this reason we recommend byte operations to the relocation register. Byte instructions may also be used for the other registers in the control block and will eliminate half of the bus cycles required if a word operation had been specified. Byte operations are only valid on even addresses though, and are undefined on odd addresses.

A.2 Peripheral Control Block Registers

Each of the integrated peripherals' control and status registers are located at a fixed location above the programmed base location of the peripheral control block. There are many locations within the peripheral control block which are not assigned to any peripheral. If a write is made to any of these locations, the bus cycle will be run, but the value will not be stored in any internal location. This means that if a subsequent read is made to the same location, the value written will not be read back.

The processor will run an external bus cycle for any memory or I/O cycle which accesses a location within the integrated control block. This means that the address, data, and control information will be driven on the 80186 external pins just as if a "normal" bus cycle had been run. Any information returned by an external device will be ignored, however, even if the access was to a location which does not correspond to any of the integrated peripheral control registers. The above is also true for the 80188, except that the word access made to the integrated registers will be performed in a single bus cycle internally, while externally, the BIU runs two bus cycles.

The processor internally generates a ready signal whenever any of the integrated peripherals are accessed; thus any external ready signals are ignored whenever an access is made to any location within the integrated peripheral register control block. This ready will also be returned if an access is made to a location within the 256 byte area of the peripheral control block which does not correspond to any integrated peripheral control register. The processor will insert 0 wait states to any access within the integrated peripheral control block except for accesses to the timer registers. ANY access to the timer control and counting registers will incur 1 wait state. This wait state is required to properly multiplex processor and counter element accesses to the timer control registers.

All accesses made to the integrated peripheral control block will be WORD accesses. Any write to the integrated registers will modify all 16 bits of the register, whether the opcode specified a byte write or a word write. A byte read from an even location should cause no problems, but the data returned when a byte read is performed from an odd address within the peripheral control block is undefined. This is true both for the 80186 AND the 80188. As stated above, even though the 80188 has an external 8 bit data bus, internally it is still a 16 bit machine. Thus, the word accesses performed to the integrated registers by the 80188 will each occur in a single bus cycle internally while externally the BIU runs two bus cycles.

APPENDIX B
80186 SYNCHRONIZATION INFORMATION

Many input signals to the 80186 are asynchronous, that is, a specified set up or hold time is not required to insure proper functioning of the device. Associated with each of these inputs is a synchronizer which samples this external asynchronous signal, and synchronizes it to the internal 80186 clock.

B.1 WHY SYNCHRONIZERS ARE REQUIRED

Every data latch requires a certain set up and hold time in order to operate properly. At a certain window within the specified set up and hold time, the part will actually try to latch the data. If the input makes a transition within this window, the output will not attain a stable state within the given output delay time. The size of this sampling window is typically much smaller than the actual window specified by the data sheet, however part to part variation could move this window around within the specified window in the data sheet.

Even if the input to a data latch makes a transition while a data latch is attempting to latch this input, the output of the latch will attain a stable state after a certain amount of time, typically much longer than the normal strobe to output delay time. Figure B-1 shows a normal input to output strobed transition and one in which the input signal makes a transition during the latch's sample window. In order to synchronize an asynchronous signal, all one needs to do is to sample the signal into one data latch, wait a certain amount of time, then latch it into a second data latch. Since the time between the strobe into the first data latch and the strobe into the second data latch allows the first data latch to attain a steady state (or to resolve the asynchronous signal), the second data latch will be presented with an input signal which satisfies any set up and hold time requirements it may have.

Thus, the output of this second latch is a synchronous signal with respect to its strobe input.

A synchronization failure can occur if the synchronizer fails to resolve the asynchronous transition within the time between the two latch's strobe signals. The rate of failure is determined by the actual size of the sampling window of the data latch, and by the amount of time between the strobe signals of the two latches. Obviously, as the sampling window gets smaller, the number of times an asynchronous transition will occur during the sampling window will drop. In addition, however, a smaller sampling window is also indicative of a faster resolution time for an input transition which manages to fall within the sampling window.

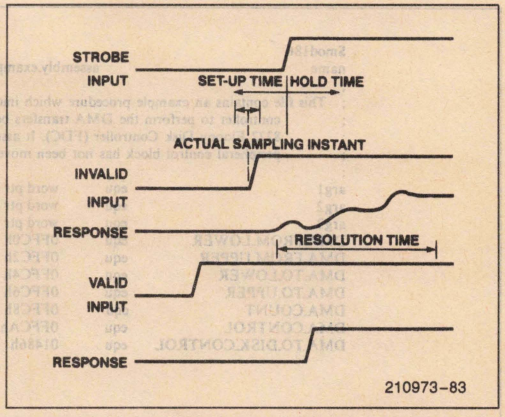


Figure B-1. Valid and Invalid Latch Input Transitions and Responses

B.2 80186 SYNCHRONIZERS

The 80186 contains synchronizers on the $\overline{\text{RES}}$, $\overline{\text{TEST}}$, TmrIn0–1, DRQ0–1, NMI, INT0–3, ARDY, and HOLD input lines. Each of these synchronizers use the two stage synchronization technique described above (with some minor modifications for the ARDY line, see section 3.1.6). The sampling window of the latches is designed to be in the tens of pico-seconds, and should allow operation of the synchronizers with a mean time between failures of over 30 years assuming continuous operation.

APPENDIX C
80186 EXAMPLE DMA INTERFACE CODE

```
$mod186
name                    assembly_example_80186_DMA_support
;
;  This file contains an example procedure which initializes the 80186 DMA
;       controller to perform the DMA transfers between the 80186 system and the
;       8272 Floppy Disk Controller (FDC). It assumes that the 80186
;       peripheral control block has not been moved from its reset location.
;
arg1                    equ     word ptr [BP + 4]
arg2                    equ     word ptr [BP + 6]
arg3                    equ     word ptr [BP + 8]
DMA_FROM_LOWER          equ     0FFC0h          ;  DMA register locations
DMA_FROM_UPPER          equ     0FFC2h
DMA_TO_LOWER            equ     0FFC4h
DMA_TO_UPPER            equ     0FFC6h
DMA_COUNT               equ     0FFC8h
DMA_CONTROL             equ     0FFCAh
DMA_TO_DISK_CONTROL     equ     01486h          ;  destination synchronization
                                                ;  source to memory, incremented
                                                ;  destination to I/O
                                                ;  no terminal count
                                                ;  byte transfers
DMA_FROM_DISK_CONTROL   equ     0A046h          ;  source synchronization
                                                ;  source to I/O
                                                ;  destination to memory, incr
                                                ;  no terminal count
                                                ;  byte transfers
FDC_DMA                 equ     6B8h            ;  FDC DMA address
FDC_DATA                equ     688h            ;  FDC data register
FDC_STATUS              equ     680h            ;  FDC status register

cgroup                  group   code
code                    segment                 public 'code'
                        public  set_dma_
                        assume  cs:cgroup
;
;  set_dma (offset,to) programs the DMA channel to point one side to the
;       disk DMA address, and the other to memory pointed to by ds:offset. If
;       'to' = 0 then will be a transfer from disk to memory; if
;       'to' = 1 then will be a transfer from memory to disk. The parameters to
;       the routine are passed on the stack.
;
set_dma_                proc    near
                        enter   0,0             ;  set stack addressability
                        push    AX              ;  save registers used
                        push    BX
                        push    DX
                        test    arg2,1          ;  check to see direction of
                                                ;  transfer
                        jz      from_disk
;  performing a transfer from memory to the disk controller
;
                        mov     AX,DS           ;  get the segment value
                        rol     AX,4            ;  gen the upper 4 bits of the
                                                ;  physical address in the lower 4
                                                ;  bits of the register
```

210973-84

```
                        mov     BX,AX                   ; save the result...
                        mov     DX,DMA_FROM_UPPER       ; prgm the upper 4 bits of the
                        out     DX,AX                   ; DMA source register
                        and     AX,0FFF0h               ; form the lower 16 bits of the
                                                        ; physical address
                        add     AX,arg1                 ; add the offset
                        mov     DX,DMA_FROM_LOWER       ; prgm the lower 16 bits of the
                        out     DX,AX                   ; DMA source register
                        jnc     no_carry_from           ; check for carry out of addition
                        inc     BX                      ; if carry out, then need to adj
                        mov     AX,BX                   ; the upper 4 bits of the pointer
                        mov     DX,DMA_FROM_UPPER
                        out     DX,AX
no_carry_from:
                        mov     AX,FDC_DMA              ; prgm the low 16 bits of the DMA
                        mov     DX,DMA_TO_LOWER         ; destination register
                        out     DX,AX
                        xor     AX,AX                   ; zero the up 4 bits of the DMA
                        mov     DX,DMA_TO_UPPER         ; destination register
                        out     DX,AX
                        mov     AX,DMA_TO_DISK_CONTROL; prgm the DMA ctl reg
                        mov     DX,DMA_CONTROL          ; note: DMA may begin immediatly
                        out     DX,AX                   ; after this word is output
                        pop     DX
                        pop     BX
                        pop     AX
                        leave
                        ret
from_disk:
;
;   performing a transfer from the disk to memory
;
                        mov     AX,DS
                        rol     AX,4
                        mov     DX,DMA_TO_UPPER
                        out     DX,AX
                        mov     BX,AX
                        and     AX,0FFF0h
                        add     AX,arg1
                        mov     DX,DMA_TO_LOWER
                        out     DX,AX
                        jnc     no_carry_to
                        inc     BX
                        mov     AX,BX
                        mov     DX,DMA_TO_UPPER
                        out     DX,AX
no_carry_to:
                        mov     AX,FDC_DMA

                        mov     DX,DMA_FROM_LOWER
                        out     DX,AX
                        xor     AX,AX
                        mov     DX,DMA_FROM_UPPER
                        out     DX,AX
                        mov     AX,DMA_FROM_DISK_CONTROL
                        mov     DX,DMA_CONTROL                                  210973-85
                        out     DX,AX
                        pop     DX
                        pop     BX
                        pop     AX
                        leave
                        ret
set_dma_                endp

code                    ends
                        end
```

210973-86

APPENDIX D
80186 EXAMPLE TIMER INTERFACE CODE

```
$mod186
name                    example_80186_timer_code
;
;  this file contains example 80186 timer routines. The first routine
;       sets up the timer and interrupt controller to cause the timer
;       to generate an interrupt every 10 milliseconds, and to service
;       interrupt to implement a real time clock. Timer 2 is used in
;       this example because no input or output signals are required.
;       The code example assumes that the peripheral control block has
;       not been moved from its reset location (FF00-FFFF in I/O space).
;
arg1                    equ     word ptr [BP + 4]
arg2                    equ     word ptr [BP + 6]
arg3                    equ     word ptr [BP + 8]
timer_2int              equ     19              ;  timer 2 has vector type 19
timer_2control          equ     0FF66h
timer_2max_ctl          equ     0FF62h
timer_int_ctl           equ     0FF32h          ;  interrupt controller regs
eoi_register            equ     0FF22h
interrupt_stat          equ     0FF30h

data                    segment                 public 'data'
                        public  hour_,minute_,second_,msec_
msec_                   db      ?
hour_                   db      ?
minute_                 db      ?
second_                 db      ?
data                    ends

cgroup                  group   code
dgroup                  group   data

code                    segment                 public 'code'
                        public  set_time_
                        assume  cs:code,ds:dgroup
;
;  set_time(hour,minute,second) sets the time variables, initializes the
;       80186 timer2 to provide interrupts every 10 milliseconds, and
;       programs the interrupt vector for timer 2
;
set_time_               proc    near
                        enter   0,0             ;  set stack addressability
                        push    AX              ;  save registers used
                        push    DX
                        push    SI
                        push    DS

                        xor     AX,AX           ;  set the interrupt vector
                                                ;  the timers have unique
                                                ;  interrupt
                                                ;  vectors even though they share
                                                ;  the same control register
                        mov     DS,AX

                        mov     SI,4 * timer2_int
```

210973-87

```
                        mov     DS:[SI],offset timer_2.interrupt_routine
                        inc     SI
                        inc     SI
                        mov     DS:[SI],CS
                        pop     DS

                        mov     AX,arg1                 ; set the time values
                        mov     hour_,AL
                        mov     AX,arg2
                        mov     minute_,AL
                        mov     AX,arg3
                        mov     second_,AL
                        mov     msec_,0

                        mov     DX,timer2_max_ctl       ; set the max count value
                        mov     AX,20000                ; 10 ms / 500 ns (timer 2 counts
                                                        ; at 1/4 the CPU clock rate)
                        out     DX,AX
                        mov     DX,timer2_control       ; set the control word
                        mov     AX,1111000000000001b    ; enable counting
                                                        ; generate interrupts on TC
                                                        ; continuous counting
                        out     DX,AX

                        mov     DX,timer_int_ctl        ; set up the interrupt controller
                        mov     AX,0000b                ; unmask interrupts
                                                        ; highest priority interrupt
                        out     DX,AX
                        sti                             ; enable processor interrupts

                        pop     SI
                        pop     DX
                        pop     AX
                        leave
                        ret
set_time_               endp

timer2_interrupt_routine proc   far
                        push    AX
                        push    DX

                        cmp     msec_,99                ; see if one second has passed
                        jae     bump_second             ; if above or equal...
                        inc     msec_
                        jmp     reset_int_ctl
bump_second:
                        mov     msec_,0                 ; reset millisecond
                        cmp     second_,59              ; see if one minute has passed
                        jae     bump_minute
                        inc     second_
                        jmp     reset_int_ctl
bump_minute:
                        mov     second_,0
                        cmp     minute_,59              ; see if one hour has passed
                        jae     bump_hour
                        inc     minute_
                        jmp     reset_int_ctl                                   210973-88
                        pop     AX
                        pop     DX
                        ret
set_rmx_                endp
code                    ends
                        end
```

210973-89

```
bump_hour:
                        mov     minute_,0
                        cmp     hour_,12        ; see if 12 hours have passed
                        jae     reset_hour
                        inc     hour_
                        jmp     reset_int_ctl
reset_hour:
                        mov     hour_,1
reset_int_ctl:

                        mov     DX,eoi_register
                        mov     AX,8000h        ; non-specific end of interrupt
                        out     DX,AX

                        pop     DX
                        pop     AX
                        iret
timer2_interrupt_routine endp
code                    ends
                        end

$mod186
name                    example_80186_baud_code
;
; this file contains example 80186 timer routines. The second routine
;       sets up the timer as a baud rate generator. In this mode,
;       Timer 1 is used to continually output pulses with a period of
;       6.5 usec for use with a serial controller at 9600 baud
;       programmed in divide by 16 mode (the actual period required
;       for 9600 baud is 6.51 usec). This assumes that the 80186 is
;       running at 8 MHz. The code example also assumes that the
;       peripheral control block has not been moved from its reset
;       location (FF00-FFFF in I/O space).
;
timer1_control          equ     0FF5Eh
timer1_max_cnt          equ     0FF5Ah

code                    segment                 public 'code'
                        assume  cs:code
;
; set_baud() initializes the 80186 timer1 as a baud rate generator for
;       a serial port running at 9600 baud
;
set_baud_               proc    near
                        push    AX              ; save registers used
                        push    DX

                        mov     DX,timer1_max_cnt ; set the max count value
                        mov     AX,13           ; 500ns * 13 = 6.5 usec
                        out     DX,AX
                        mov     DX,timer1_control ; set the control word
                        mov     AX,1100000000000001b ; enable counting
                                                ; no interrupt on TC
                                                ; continuous counting
                                                ; single max count register
                        out     DX,AX

                        pop     DX
                        pop     AX
```

210973-90

```
                        ret
set_baud_               endp
code                    ends
                        end

$mod186
name                    example_80186_count_code
;
;  this file contains example 80186 timer routines. The third routine
;       sets up the timer as an external event counter. In this mode,
;       Timer 1 is used to count transitions on its input pin. After
;       the timer has been set up by the routine, the number of
;       events counted can be directly read from the timer count
;       register at location FF58H in I/O space. The timer will
;       count a maximum of 65535 timer events before wrapping
;       around to zero. This code example also assumes that the
;       peripheral control block has not been moved from its reset
;       location (FF00-FFFF in I/O space).
;
timer1_control          equ     0FF5Eh
timer1_max_cnt          equ     0FF5Ah
timer1_cnt_reg          equ     0FF58H

code                    segment                         public 'code'
                        assume  cs:code
;
;  set_count() initializes the 80186 timer1 as an event counter
;
set_count_              proc    near
                        push    AX                      ;  save registers used
                        push    DX

                        mov     DX,timer1_max_cnt       ;  set the max count value
                        mov     AX,0                    ;  allows the timer to count
                                                        ;  all the way to FFFFH
                        out     DX,AX
                        mov     DX,timer1_control       ;  set the control word
                        mov     AX,1100000000000101b    ;  enable counting
                                                        ;  no interrupt on TC
                                                        ;  continuous counting
                                                        ;  single max count register
                                                        ;  external clocking
                        out     DX,AX

                        xor     AX,AX                   ;  zero AX
                        mov     DX,timer1_cnt_reg       ;  and zero the count in the timer
                        out     DX,AX                   ;  count register

                        pop     DX
                        pop     AX
                        ret

set_count_              endp
code                    ends
                        end
```

210973-91

APPENDIX E
80186 EXAMPLE INTERRUPT CONTROLLER INTERFACE CODE

```
$mod186
name                    example_80186_interrupt_code
;
;  This routine configures the 80186 interrupt controller to provide
;       two cascaded interrupt inputs (through an external 8259A
;       interrupt controller on pins INT0/INT2) and two direct
;       interrupt inputs (on pins INT1 and INT3). The default priority
;       levels are used. Because of this, the priority level programmed
;       into the control register is set the 111, the level all
;       interrupts are programmed to at reset.
;
int0_control            equ     0FF38H
int_mask                equ     0FF28H
;
code                    segment                         public 'code'
                        assume  CS:code
set_int_                proc    near
                        push    DX
                        push    AX

                        mov     AX,0100111B             ;  cascade mode
                                                        ;  interrupt unmasked
                        mov     DX,int0_control
                        out     DX,AX

                        mov     AX,01001101B            ;  now unmask the other external
                                                        ;  interrupts
                        mov     DX,int_mask
                        out     DX,AX
                        pop     AX
                        pop     DX
                        ret
set_int_                endp
code                    ends
                        end
$mod186
name                    example_80186_interrupt_code
;
;  This routine configures the 80186 interrupt controller into iRMX 86
;       mode. This code does not initialize any of the 80186
;       integrated peripheral control registers, nor does it initialize
;       the external 8259A or 80130 interrupt controller.
;
relocation_reg          equ     0FFFEH
;
code                    segment                         public 'code'
                        assume  CS:code
set_rmx_                proc    near
                        push    DX
                        push    AX

                        mov     DX,relocation_reg
                        in      AX,DX                   ;  read old contents of register
                        or      AX,0100000000000000B    ;  set the RMX mode bit
                        out     DX,AX
```

210973-92

APPENDIX F
80186/8086 EXAMPLE SYSTEM INITIALIZATION CODE

```
name            example_80186_system_init
;
;  This file contains a system initialization routine for the 80186
;       or the 8086. The code determines whether it is running on
;       an 80186 or an 8086, and if it is running on an 80186, it
;       initializes the integrated chip select registers.
;
restart         segment at                      0FFFFh
;
;  This is the processor reset address at 0FFFF0H
;
                org     0
                jmp     far ptr initialize
restart         ends
;
                extrn   monitor:far
init_hw         segment at                      0FFF0h
                assume  CS:init_hw
;
;  This segment initializes the chip selects. It must be located in the
;       top 1K to insure that the ROM remains selected in the 80186
;       system until the proper size of the select area can be programmed.
;
UMCS_reg        equ     0FFA0H          ;  chip select register locations
LMCS_reg        equ     0FFA2H
PACS_reg        equ     0FFA4H
MPCS_reg        equ     0FFA8H
UMCS_value      equ     0F038H          ;  64K, no wait states
LMCS_value      equ     07F8H           ;  32K, no wait states
PACS_value      equ     007EH           ;  peripheral base at 400H, 2 ws
MPCS_value      equ     81B8H           ;  PCS5 and 6 supplies,
                                        ;  peripherals in I/O space
initialize      proc    far
                mov     AX,2            ;  determine if this is an
                mov     CL,33           ;  8086 or an 80186 (checks
                shr     AX,CL           ;  to see if the multiple bit
                test    AX,1            ;  shift value was ANDed)
                jz      not_80186

                mov     DX,UMCS_reg     ;  program the UMCS register
                mov     AX,UMCS_value
                out     DX,AX

                mov     DX,LMCS_reg     ;  program the LMCS register
                mov     AX,LMCS_value
                out     DX,AX

                mov     DX,PACS_reg     ;  set up the peripheral chip
                                        ;  selects (note the mid-range
                                        ;  memory chip selects are not
                                        ;  needed in this system, and
                                        ;  are thus not initialized
                mov     AX,PACS_value
                out     DX,AX                                   210973-93
                mov     DX,MPCS_reg
                mov     AX,MPCS_value
                out     DX,AX
;
;  Now that the chip selects are all set up, the main program of the
;       computer may be executed.
;
not_80186:
                jmp     far ptr monitor
initialize      endp
init_hw         ends
                end
```

210973-94

APPENDIX G
80186 WAIT STATE PERFORMANCE

Because the 80186 contains separate bus interface and execution units, the actual performance of the processor will not degrade at a constant rate as wait states are added to the memory cycle time from the processor. The actual rate of performance degradation will depend on the type and mix of instructions actually encountered in the user's program.

Shown below are two 80186 assembly language programs, and the actual execution time for the two programs as wait states are added to the memory system of the processor. These programs show the two extremes to which wait states will or will not affect system performance as wait states are introduced.

Program 1 is very memory intensive. It performs many memory reads and writes using the more extensive memory addressing modes of the processor (which also take a greater number of bytes in the opcode for the instruction). As a result, the execution unit must constantly wait for the bus interface unit to fetch and perform the memory cycles to allow it to continue. Thus, the execution time of this type of routine will grow quickly as wait states are added, since the execution time is almost totally limited to the speed at which the processor can run bus cycles.

Note also that this program execution time calculated by merely summing up the number of clock cycles given in the data sheet will typically be less than the actual number of clock cycles actually required to run the program. This is because the numbers quoted in the data sheet assume that the opcode bytes have been prefetched and reside in the 80186 prefetch queue for immediate access by the execution unit. If the execution unit cannot access the opcode bytes immediately upon request, dead clock cycles will be inserted in which the execution unit will remain idle, thus increasing the number of clock cycles required to complete execution of the program.

On the other hand, program 2 is more CPU intensive. It performs many integer multiplies, during which time the bus interface unit can fill up the instruction prefetch queue in parallel with the execution unit performing the multiply. In this program, the bus interface unit can perform bus operations faster than the execution unit actually requires them to be run. In this case, the performance degradation is much less as wait states are added to the memory interface. The execution time of this program is closer to the number of clock cycles calculated by adding the number of cycles per instruction because the execution unit does not have to wait for the bus interface unit to place an opcode byte in the prefetch queue as often. Thus, fewer clock cycles are wasted by the execution unit laying idle for want of instructions. Table G-1 lists the execution times measured for these two programs as wait states were introduced with the 80186 running at 8 MHz.

Table G-1

# of Wait States	Program 1		Program 2	
	Exec Time (μsec)	Perf Degr	Exec Time (μsec)	Perf Degr
0	505		294	
1	595	18%	311	6%
2	669	12%	337	8%
3	752	12%	347	3%

```
$mod186
name                    example_wait_state_performance
;
; This file contains two programs which demonstrate the 80186 performance
;       degradation as wait states are inserted. Program 1 performs a
;       transformation between two types of characters sets, then copies
;       the transformed characters back to the original buffer (which is 64
;       bytes long. Program 2 performs the same type of transformation, however
;       instead of performing a table lookup, it multiplies each number in the
;       original 32 word buffer by a constant (3, note the use of the integer
;       immediate multiply instruction). Program "nothing" is used to measure
;       the call and return times from the driver program only.
;
cgroup                  group   code
dgroup                  group   data
data                    segment                 public 'data'
```

210973-95

```
t_table         db      256 dup (?)
t_string        db      64 dup (?)
m_array         dw      32 dup (?)
data            ends

code            segment                         public 'code'
                assume  CS:cgroup,DS:dgroup
                public  bench_1,bench_2,nothing_,wait_state_,set_timer_
bench_1         proc    near
                push    SI                      ; save registers used
                push    CX
                push    BX
                push    AX

                mov     CX,64                   ; translate 64 bytes
                mov     SI,0
                mov     BH,0

loop_back:
                mov     BL,t_string[SI]         ; get the byte
                mov     AL,t_table[BX]          ; translate byte
                mov     t_string[SI],AL         ; and store it
                inc     SI                      ; increment index
                loop    loop_back               ; do the next byte

                pop     AX
                pop     BX
                pop     CX
                pop     SI
                ret
bench_1         endp

bench_2         proc    near
                push    AX                      ; save registers used
                push    SI
                push    CX

                mov     CX,32                   ; multiply 32 numbers
                mov     SI,offset m_array

loop_back_2:
                imul    AX,word ptr [SI],3      ; immediate multiply
                mov     word ptr [SI],AX
                inc     SI
                inc     SI
                loop    loop_back_2

                pop     CX
                pop     SI
                pop     AX
                ret
bench_2_        endp
```

210973-96

```
nothing_                proc    near
                        ret
nothing_                endp
;
;  wait_state(n) sets the 80186 LMCS register to the number of wait states
;          (0 to 3) indicated by the parameter n (which is passed on the stack).
;          No other bits of the LMCS register are modified.
;
wait_state_             proc    near
                        enter   0,0             ;  set up stack frame
                        push    AX              ;  save registers used
                        push    BX
                        push    DX

                        mov     BX,word ptr [BP + 4]    ;  get argument
                        mov     DX,0FFA2h       ;  get current LMCS register
contents
                        in      AX,DX

                        and     AX,0FFFCh       ;  and off existing ready bits
                        and     BX,3            ;  insure ws count is good
                        or      AX,BX           ;  adjust the ready bits
                        out     DX,AX           ;  and write to LMCS

                        pop     DX
                        pop     BX
                        pop     AX
                        leave                   ;  tear down stack frame
                        ret
wait_state_             endp
;
;  set_timer() initializes the 80186 timers to count microseconds. Timer 2
;          is set up as a prescaler to timer 0, the microsecond count can be read
;          directly out of the timer 0 count register at location FF50H in I/O
;          space.
;
set_timer_              proc    near
                        push    AX
                        push    DX

                        mov     DX,0ff66h       ;  stop timer 2
                        mov     AX,4000h
                        out     DX,AX

                        mov     DX,0ff50h       ;  clear timer 0 count
                        mov     AX,0
                        out     DX,AX

                        mov     DX,0ff52h       ;  timer 0 counts up to 65535
                        mov     AX,0
                        out     DX,AX
```

210973-97

```
                        mov     DX,0ff56h       ; enable timer 0
                        mov     AX,0c009h
                        out     DX,AX

                        mov     DX,0ff60h       ; clear timer 2 count
                        mov     AX,0
                        out     DX,AX

                        mov     DX,0ff62h       ; set maximum count of timer 2
                        mov     AX,2
                        out     DX,AX

                        mov     DX,0ff66h       ; re-enable timer 2
                        mov     AX,0c001h
                        out     DX,AX

                        pop     DX
                        pop     AX
                        ret
set_timer_              endp
code                    ends
                        end
```

210973-98

APPENDIX H
80186 NEW INSTRUCTIONS

The 80186 performs many additional instructions to those of the 8086. These instructions appear shaded in the instruction set summary at the back of the 80186 data sheet. This appendix explains the operation of these new instructions. In order to use these new instructions with the 8086/186 assembler, the "$mod186" switch must be given to the assembler. This can be done by placing the line: "$mod186" at the beginning of the assembly language file.

PUSH IMMEDIATE

This instruction allows immediate data to be pushed onto the processor stack. The data can be either an immediate byte or an immediate word. If the data is a byte, it will be sign extended to a word before it is pushed onto the stack (since all stack operations are word operations).

PUSHA, POPA

These instructions allow all of the general purpose 80186 registers to be saved on the stack, or restored from the stack. The registers saved by this instruction (in the order they are pushed onto the stack) are AX, CX, DX, BX, SP, BP, SI, and DI. The SP value pushed onto the stack is the value of the register before the first PUSH (AX) is performed; the value popped for the SP register is ignored.

This instruction does not save any of the segment registers (CS, DC, SS, ES), the instruction pointer (IP), the flag register, or any of the integrated peripheral registers.

IMUL BY AN IMMEDIATE VALUE

This instruction allows a value to be multiplied by an immediate value. The result of this operation is 16 bits long. One operand for this instruction is obtained using one of the 80186 addressing modes (meaning it can be in a register or in memory). The immediate value can be either a byte or a word, but will be sign extended if it is a byte. The 16-bit result of the multiplication can be placed in any of the 80186 general purpose or pointer registers.

This instruction requires three operands: the register in which the result is to be placed, the immediate value, and the second operand. Again, this second operand can be any of the 80186 general purpose registers or a specified memory location.

SHIFTS/ROTATES BY AN IMMEDIATE VALUE

The 80186 can perform multiple bit shifts or rotates where the number of bits to be shifted is specified by an immediate value. This is different from the 8086, where only a single bit shift can be performed, or a multiple shift can be performed where the number of bits to be shifted is specified in the CL register.

All of the shift/rotate instructions of the 80186 allow the number of bits shifted to be specified by an immediate value. Like all multiple bit shift operations performed by the 80186, the number of bits shifted is the number of bits specified modulus 32 (i.e., the maximum number of bits shifted by the 80186 multiple bit shifts is 31).

These instructions require two operands: the operand to be shifted (which may be a register or a memory location specified by any of the 80186 addressing modes) and the number of bits to be shifted.

BLOCK INPUT/OUTPUT

The 80186 adds two new input/output instructions: INS and OUTS. These instructions perform block input or output operations. They operate similarly to the string move instructions of the processor.

The INS instruction performs block input from an I/O port to memory. The I/O address is specified by the DX register; the memory location is pointed to by the DI register. After the operation is performed, the DI register is adjusted by 1 (if a byte input is specified) or by 2 (if a word input is specified). The adjustment is either an increment or a decrement, as determined by the Direction bit in the flag register of the processor. The ES segment register is used for memory addressing, and cannot be overridden. When preceded by a REPeat prefix, this instruction allows blocks of data to be moved from an I/O address to a block of memory. Note that the I/O address in the DX register is not modified by this operation.

The OUTS instruction performs block output from memory to an I/O port. The I/O address is specified by the DX register; the memory location is pointed to by the SI register. After the operation is performed, the SI register is adjusted by 1 (if a byte output is specified) or by 2 (if a word output is specified). The adjustment is either an increment or a decrement, as determined by the Direction bit in the flag register of the processor. The DS segment register is used for memory addressing, but can be overridden by using a segment override prefix. When preceded by a REPeat prefix, this instruction allows blocks of data to be moved from a block of memory to an I/O address. Again note that the I/O address in the DX register is not modified by this operation.

Like the string move instruction, these two instructions require two operands to specify whether word or byte operations are to take place. Additionally, this determination can be supplied by the mnemonic itself by adding a "B" or "W" to the basic mnemonic, for example:

```
INSB          ;perform byte input

REP OUTSW     ;perform word block output
```

BOUND

The 80186 supplies a BOUND instruction to facilitate bound checking of arrays. In this instruction, the calculated index into the array is placed in one of the general purpose registers of the 80186. Located in two adjacent word memory locations are the lower and upper bounds for the array index. The BOUND instruction compares the register contents to the memory locations, and if the value in the register is not between the values in the memory locations, an interrupt type 5 is generated. The comparisons performed are SIGNED comparisons. A register value equal to either the upper bound or the lower bound will not cause an interrupt.

This instruction requires two arguments: the register in which the calculated array index is placed, and the word memory location which contains the lower bound of the array (which can be specified by any of the 80186 memory addressing modes). The memory location containing the upper bound of the array must follow immediately the memory location containing the lower bound of the array.

ENTER AND LEAVE

The 80186 contains two instructions which are used to build and tear down stack frames of higher level, block structured languages. The instruction used to build these stack frames is the ENTER instruction. The algorithm for this instruction is:

```
PUSH BP              /*save the previous  frame
                     pointer*/
if level=0 then
   BP:=SP;
else   templ:=SP;/*save current frame  pointer
                  */
  temp2:= level - 1;
  do while temp2>0/*copy down previous  level
                   frame*/
    BP:= BP - 2;   /*pointers*/
    PUSH [BP];
  BP:=templ;
  PUSH BP;         /*put current level  frame
                   pointer*/

/*in the save area*/
SP:=SP - disp;     /*create space on the  stack
                   for*/

/*local variables*/
```

Figure H-1 shows the layout of the stack before and after this operation.

This instruction requires two operands: the first value (disp) specifies the number of bytes the local variables of this routine require. This is an unsigned value and can be as large as 65535. The second value (level) is an unsigned value which specifies the level of the procedure. It can be as great as 255.

The 80186 includes the LEAVE instruction to tear down stack frames built up by the ENTER instruction. As can be seen from the layout of the stack left by the ENTER instruction, this involves only moving the contents of the BP register to the SP register, and popping the old BP value from the stack.

Neither the ENTER nor the LEAVE instructions save any of the 80186 general purpose registers. If they must be saved, this must be done in addition to the ENTER and the LEAVE. In addition, the LEAVE instruction does not perform a return from a subroutine. If this is desired, the LEAVE instruction must be explicitly followed by the RET instruction.

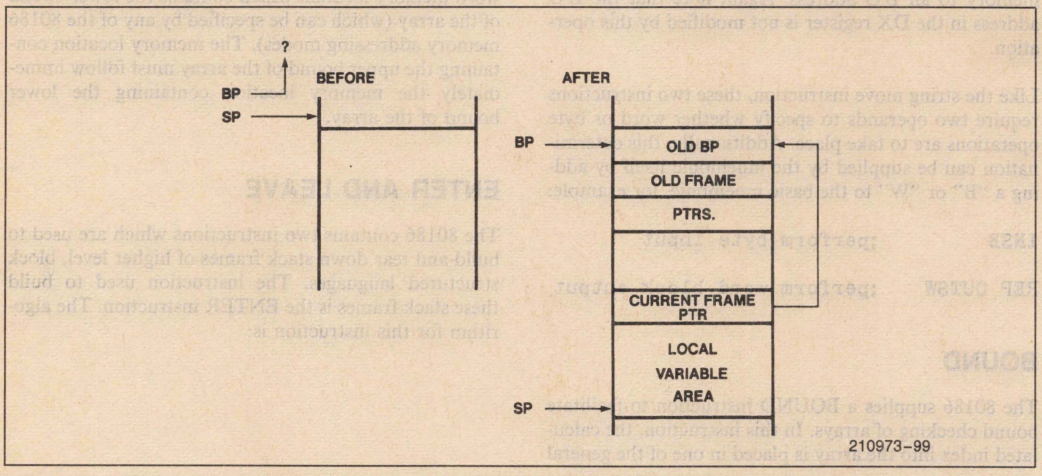


Figure H-1. ENTER Instruction Stack Frame

APPENDIX I
80186/80188 DIFFERENCES

The 80188 is exactly like the 80186, except it has an 8 bit external bus. It shares the same execution unit, timers, peripheral control block, interrupt controller, chip select, and DMA logic. The differences between the two caused by the narrower data bus are:

- The 80188 has a 4 byte prefetch queue, rather than the 6 byte prefetch queue present on the 80186. The reason for this is since the 80188 fetches opcodes one byte at a time, the number of bus cycles required to fill the smaller queue of the 80188 is actually greater than the number of bus cycles required to fill the queue of the 80186. As a result, a smaller queue is required to prevent an inordinate number of bus cycles being wasted by prefetching opcodes to be discarded during a jump.
- AD8–AD15 on the 80186 are transformed to A8–A15 on the 80188. Valid address information is present on these lines throughout the bus cycle of the 80188. Valid address information is not guaranteed on these lines during idle T states.
- $\overline{BHE}$/S7 is always defined HIGH by the 80188, since the upper half of the data bus is non-existent.
- The DMA controller of the 80188 only performs byte transfers. The $\overline{B}$/W bit in the DMA control word is ignored.
- Execution times for many memory access instructions are increased because the memory access must be funnelled through a narrower data bus. The 80188 also will be more bus limited than the 80186 (that is, the execution unit will be required to wait for the opcode information to be fetched more often) because the data bus is narrower. The execution time within the processor, however, has not changed between the 80186 and 80188.

Another important point is that the 80188 internally is a 16-bit machine. This means that any access to the integrated peripheral registers of the 80188 will be done in 16-bit chunks, NOT in 8-bit chunks. All internal peripheral registers are still 16-bits wide, and only a single read or write is required to access the registers. When a word access is made to the internal registers, the BIU will run two bus cycles externally.

Access to the control block may also be done with byte operations. Internally the full 16-bits of the AX register will be written, while externally, only one bus cycle will be executed.

intel

APPLICATION NOTE

AP-258

February 1986

High Speed Numerics with the 80186/80188 and 8087

STEVE FARRER
APPLICATIONS ENGINEER

Order Number: 231590-001

1.0 INTRODUCTION

From their introduction in 1982, the highly integrated 16-bit 80186 and its 8-bit external bus version, the 80188, have been ideal processor choices for high-performance, low-cost embedded control applications. The integrated peripheral functions and enhanced 8086 CPU of the 80186 and 80188 allow for an easy upgrade of older generation control applications to achieve higher performance while lowering the overall system cost through reduced board space, and a simplified production flow.

More and more controller applications need even higher performance in numerics, yet still require the low-cost and small form factor of the 80186 and 80188. The 8087 Numerics Data Coprocessor satisfies this need as an optional add-on component.

The 8087 Numeric Data Coprocessor is interfaced to the 80186 and 80188 through the 82188 IBC (Integrated Bus Controller). The IBC provides a highly integrated interface solution which replaces the 8288 used in 8086–8087 systems. The IBC incorporates all the necessary bus control for the 8087 while also providing the necessary logic to support the interface between the 80186/8 and the 8087.

This application note discusses the design considerations associated with using the 8087 Numeric Data Coprocessor with the 80186 and 80188. Sections two, three, and four contain an overview of the integrated circuits involved in the numerics configuration. Section five discusses the interfacing aspects between the 80186/8 and the 8087, including the role of the 82188 Integrated Bus Controller and the operation of the integrated peripherals on the 80186/8 with the 8087. Section six compares the advantages of using an 8087 Numeric Data Coprocessor over software routines written for the host processor as well as the advantage of using an 80186/8 numerics system over an 8086/8088 numerics system.

Except where noted, all future references to the 80186 will apply equally to the 80188.

2.0 OVERVIEW OF THE 80186

The 80186 and 80188 are highly integrated microprocessors which effectively combine up to 20 of the most common system components onto a single chip. The 80186 and 80188 processors are designed to provide both higher performance and a more highly integated solution to the total system.

Higher integration results from integrating system peripherals onto the microprocessor. The peripherals consist of a clock generator, an interrupt controller, a DMA controller, a counter/timer unit, a programmable wait state generator, programmable chip selects, and a bus controller. (See Figure 1.)

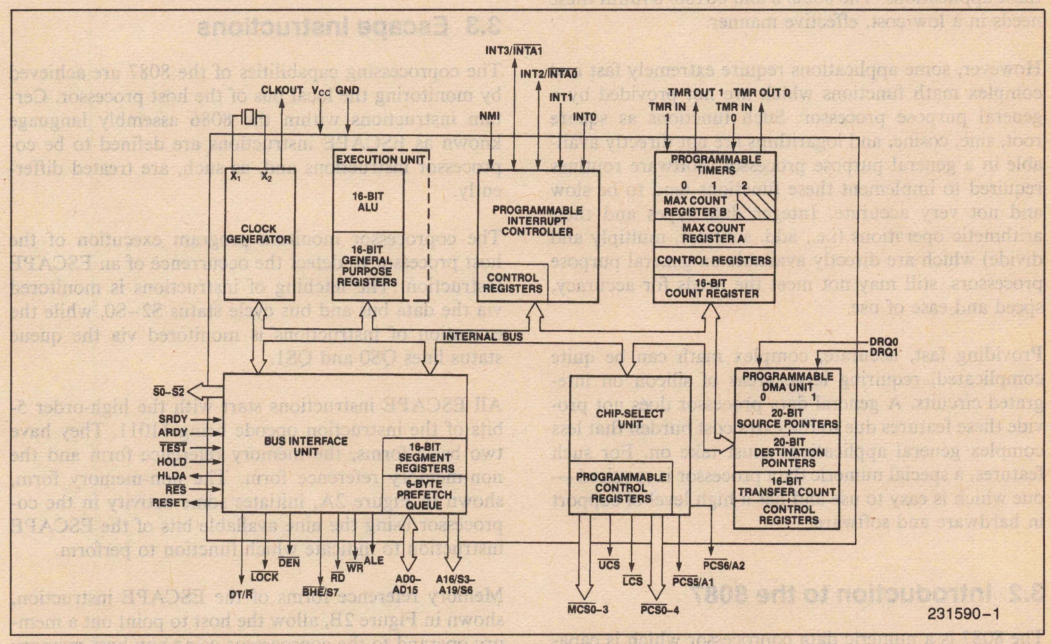


Figure 1. 80186/8 Block Diagram

Higher performance results from enhancements to both general and specific areas of the 8086 CPU, including faster effective address calculation, improvement in the execution speed of many instructions, and the inclusion of new instructions which are designed to produce optimum 80186 code.

The 80186 and 80188 are completely object code compatible with the 8086 and 8088. They have the same basic register set, memory organization, and addressing modes. The differences between the 80186 and 80188 are the same as the differences between the 8086 and 8088: the 80186 has a 16-bit architecture and 16-bit bus interface; the 80188 has a 16-bit internal architecture and an 8-bit data bus interface. The instruction execution times of the two processors differ accordingly: for each non-immediate 16-bit data read/write instruction, 4 additional clock cycles are required by the 80188.

3.0 NUMERICS OVERVIEW

3.1 The Benefits of Numeric Coprocessing

The 8086/8 and 80186/8 are general purpose microprocessors, designed for a very wide range of applications. Typically, these applications need fast, efficient data movement and general purpose control instructions. Arithmetic on data values tends to be simple in these applications. The 8086/8 and 80186/8 fulfill these needs in a low cost, effective manner.

However, some applications require extremely fast and complex math functions which are not provided by a general purpose processor. Such functions as square root, sine, cosine, and logarithms are not directly available in a general purpose processor. Software routines required to implement these functions tend to be slow and not very accurate. Integer data types and their arithmetic operations (i.e., add, subtract, multiply and divide) which are directly available on general purpose processors, still may not meet the needs for accuracy, speed and ease of use.

Providing fast, accurate, complex math can be quite complicated, requiring large areas of silicon on integrated circuits. A general data processor does not provide these features due to the extra cost burden that less complex general applications must take on. For such features, a special numeric data processor is required — one which is easy to use and has a high level of support in hardware and software.

3.2 Introduction to the 8087

The 8087 is a numeric data coprocessor which is capable of performing complex mathematical functions while the host processor (i.e. the main CPU) performs more general tasks. It supports the necessary data types and operations and allows use of all the current hardware and software support for the 8086/8 and 80186/8 microprocessors. The fact that the 8087 is a coprocessor means it is capable of operating in parallel with the host CPU, which greatly improves the processing power of the system.

The 8087 can increase the performance of floating-point calculations by 50 to 100 times, providing the performance and precision required for small business and graphics applications as well as scientific data processing.

The 8087 numeric coprocessor adds 68 floating-point instructions and eight 80-bit floating-point registers to the basic 8086 programming architecture. All the numeric instructions and data types of the 8087 are used by the programmer in the same manner as the general data types and instructions of the host.

The numeric data formats and arithmetic operations provided by the 8087 support the proposed IEEE Microprocessor Floating Point Standard. All of the proposed IEEE floating point standard algorithms, exception detection, exception handling, infinity arithmetic and rounding controls are implemented. The IEEE standard makes it easier to use floating point and helps to avoid common problems that are inherent to floating point.

3.3 Escape Instructions

The coprocessing capabilities of the 8087 are achieved by monitoring the local bus of the host processor. Certain instructions within the 8086 assembly language known as ESCAPE instructions are defined to be coprocessor instructions and, as such, are treated differently.

The coprocessor monitors program execution of the host processor to detect the occurrence of an ESCAPE instruction. The fetching of instructions is monitored via the data bus and bus cycle status S2–S0, while the execution of instructions is monitored via the queue status lines QS0 and QS1.

All ESCAPE instructions start with the high-order 5-bits of the instruction opcode being 11011. They have two basic forms, the memory reference form and the non-memory reference form. The non-memory form, shown in Figure 2A, initiates some activity in the coprocessor using the nine available bits of the ESCAPE instruction to indicate which function to perform.

Memory reference forms of the ESCAPE instruction, shown in Figure 2B, allow the host to point out a memory operand to the coprocessor using any host memory

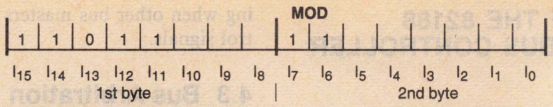


Figure 2A. Non-Memory Reference ESCAPE Instructions

addressing mode. Six bits are available in the memory reference form to identify what to do with the memory operand.

Memory reference forms of ESCAPE instructions are identified by bits 7 and 6 of the byte following the ESCAPE opcode. These two bits are the MOD field of the 8086/8 or 80186/8 effective address calculation byte. Together with the R/M field (bits 2 through 0), they determine the addressing mode and how many subsequent bytes remain in the instruction.

3.4 Host Response to Escape Instructions

The host performs one of two possible actions when encountering an ESCAPE instruction: do nothing (operation is internal to 8087) or calculate an effective address and read a word value beginning at that address (required for all LOADS and STORES). The host ignores the value of the word read and hence the cycle is referred to as a "Dummy Read Cycle." ESCAPE instructions do not change any registers in the host other than advancing the IP. If there is no coprocessor or the coprocessor ignores the ESCAPE instruction, the ESCAPE instruction is effectively a NOP to the host. Other than calculating a memory address and reading a word of memory, the host makes no other assumptions regarding coprocessor activity.

The memory reference ESCAPE instructions have two purposes: to identify a memory operand and, for certain instructions, to transfer a word from memory to the coprocessor.

3.5 Coprocessor Response to Escape Instructions

The 8087 performs basically three types of functions when encountering an ESCAPE instruction: LOAD (read from memory), STORE (write to memory), and EXECUTE (perform one of the internal 8087 math functions).

When the host executes a memory reference ESCAPE instruction intended to cause a read operation by the 8087, the host always reads the low-order word of any 8087 memory operand. The 8087 will save the address and data read. To read any subsequent words of the operand, the 8087 must become a local bus master.

When the 8087 has the local bus, it increments the 20-bit physical address it saved to address the remaining words of the operand.

When the ESCAPE instruction is intended to cause a write operation by the 8087, the 8087 will save the address but ignore the data read. Eventually, it will get control of the local bus and perform successive writes incrementing the 20-bit address after each word until the entire numeric variable has been written.

ESCAPE instructions intended to cause the execution of a coprocessor calculation do not require any bus activity. Numeric calculations work off of an internal register stack which has been initialized using a LOAD operation. The calculation takes place using one or two of the stack positions specified by the ESCAPE instruction. The result of the operation is also placed in one of the stack positions specified by the ESCAPE instruction. The result may then be returned to memory using a STORE instruction, thus allowing the host processor to access it.

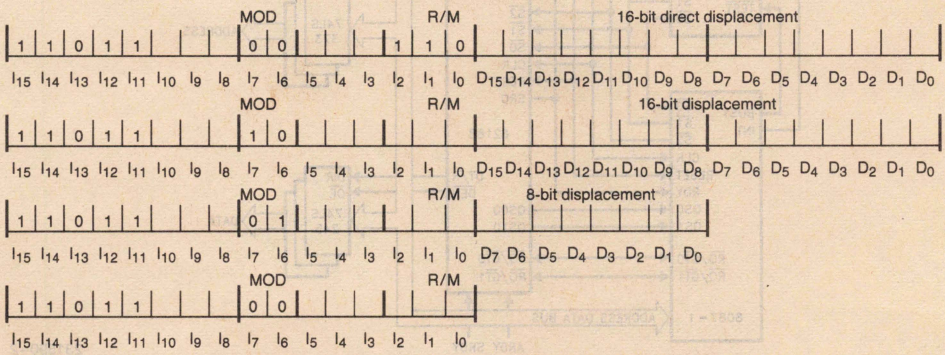


Figure 2B. Memory Reference ESCAPE Instruction Forms

4.0 OVERVIEW OF THE 82188 INTEGRATED BUS CONTROLLER

4.1 Introduction

The 82188 Integrated Bus Controller (IBC) is a highly integrated version of the 8288 Bus Controller. The IBC provides command and control timing signals for bus control and all of the necessary logic to interface the 80186 to the 8087.

4.2 Bus Control Signals

The bus command and control signals consist of $\overline{RD}$, $\overline{WR}$, $\overline{DEN}$, DT/$\overline{R}$, and ALE. The timings and levels are driven following the latching of valid signals on the status lines S0–S2. When S0–S2 change state from passive to active, the IBC begins cycling through a state machine which drives the corresponding control and command lines for the bus cycle. As with the 8288, an address enable input ($\overline{AEN}$) is present to allow tri-stating when other bus masters supply their own bus control signals.

4.3 Bus Arbitration

The IBC also has the ability to convert bus arbitration protocols of $\overline{RQ}$/$\overline{GT}$ to HOLD-HLDA. This allows the 82586 Local Area Network (LAN) Coprocessor, the 82730 Text Coprocessor, and other coprocessors using the HOLD-HLDA protocol to be interfaced to the 8086/8 as well as allowing the 80186/8 to be interfaced to the 8087. In addition to converting arbitration protocols, the IBC makes it possible to arbitrate between two bus masters using HOLD-HLDA with a third using $\overline{RQ}$/$\overline{GT}$.

4.4 Interface Logic

In addition to all the bus control and arbitration features, the IBC provides logic to connect the queue status to the 8087, a chip-select for the 8087, and the necessary READY synchronization required between the 8087 and the 80186/8.

5.0 DESIGNING THE SYSTEM

5.1 Circuit Schematics of the 80186/8–82888–8087 System

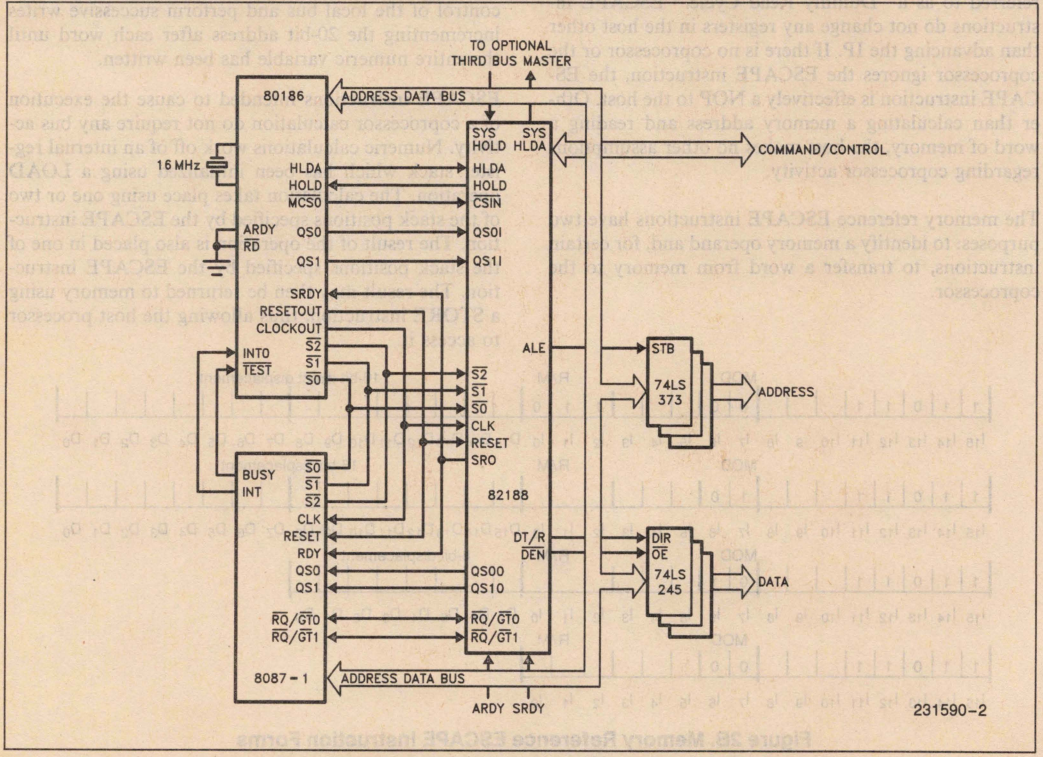


Figure 3. 80186/8–82188–8087 Circuit Diagram

5.2 Queue Status

The 8087 tracks the instruction execution of the 80186 by keeping an internal instruction queue which is identical to the processor's instruction queue. Each time the processor performs an instruction fetch, the 8087 latches the instruction into its own queue in parallel with the processor. Each time the processor removes the first byte of an instruction from the queue, the 8087 removes the byte at the top of the 8087 queue and checks to see if the byte is an ESCAPE prefix. If it is, the 8087 decodes the following bytes in parallel with the processor to determine which numeric instruction the bytes represent. If the first byte of the instruction is not an ESCAPE prefix, the 8087 discards it along with the subsequent bytes of the non-numeric instruction as the 80186 removes them from the queue for execution.

The 8087 operates its internal instruction queue by monitoring the two queue status lines from the CPU. This status information is made available by the CPU by placing it into queue status mode. This requires strapping the $\overline{RD}$ pin on the 80186 to ground. When $\overline{RD}$ is tied to ground, ALE and $\overline{WR}$ become QS0 (Queue Status #0) and QS1 (Queue Status #1) respectively.

Table 1. Queue Status Decoding

QS1	QS0	Queue Operation
0	0	No queue operation
0	1	First byte from queue
1	0	Subsequent byte from queue
1	1	Reserved

Each time the 80186 begins decoding a new instruction, the queue status lines indicate "first byte of instruction taken from the queue". This signals the 8087 to check for an ESCAPE prefix. As the remaining bytes of the instruction are removed, the queue status indicates "subsequent byte removed from queue". The 8087 uses this status to either continue decoding subsequent bytes, if the first byte was an ESCAPE prefix, or to discard the subsequent bytes if the first byte was not an ESCAPE prefix.

The QS0(ALE) and QS1($\overline{WR}$) pins of the 80186 are fed directly to the 82188 where they are latched and delayed by one-half-clock. The delayed queue status from the 82188 is then presented directly to the 8087.

The waveforms of the queue status signals are shown in Figure 4. The critical timings are the setup time into the 82188 from the 80186 and the setup and hold time into the 8087 from the 82188. The calculations for an 8 MHz system are as follows:

$.5T_{CLCL} - T_{CHQSV}$ (186 max) $.5(125\text{ ns}) - 35$	$\geq T_{QIVCL}$ (82188 min) ≥ 15 ns	;setup to 82188
$T_{CLCL} - T_{CLQOV}$ (82188 max) $(125\text{ ns}) - 50$	$\geq T_{QVCL}$ ≥ 10 ns	;setup to 8087
T_{CLQOV} (82188 min) 5	$\geq T_{CLQX}$ (8087 min) ≥ 5 ns	;hold to 8087

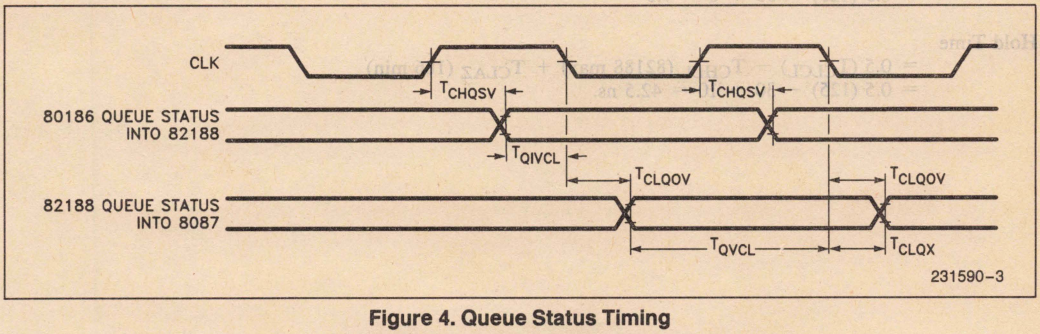


Figure 4. Queue Status Timing

5.3 Bus Control Signals

When the 80186 is in Queue Status mode, another component must generate the ALE, $\overline{RD}$, and $\overline{WR}$ signals. The 82188 provides these signals by monitoring the CPU bus cycle status ($\overline{S0}$–$\overline{S2}$). Also provided are $\overline{DEN}$ and DT/$\overline{R}$ which may be used for extra drive capability on the control bus. With the exception of ALE, all control signals on the 82188 are almost identical to their corresponding 80186 control signals. This section discusses the differences between the 80186 and the 82188 control signals for the purpose of upgrading an 80186 design to an 80186–8087 design. For original 80186–8087 designs, there is no need to compare control signal timings of the 82188 with the 80186.

5.3.1 ALE

The ALE (Address Latch Enable) signal goes active one clock phase earlier on the 80186 than on the 82188. Timing of the ALE signal on the 82188 is closer to that of the 8086 and 8288 bus controller because the bus cycle status is used to generate the ALE pulse. ALE on the 80186 goes active before the bus cycle status lines are valid.

The inactive edge of ALE occurs in the same clock phase for both the 80186 and the 82188. The setup and hold times of the 80186 address relative to the 82188 ALE signal are shown in Figure 5 and are calculated for an 8 MHz system as follows:

Setup Time

For 80186 = T_{AVCH} (186 min) + T_{CHLL} (82188 min)
= 10 + 0 = 10 ns.

For 8087 = 0.5 (T_{CLCL}) − T_{CLAV} (8087 max) + T_{CHLL} (82188 min)
= 0.5 (125) − 55 + 0 = 7.5

Hold Time

= 0.5 (T_{CLCL}) − T_{CHLL} (82188 max) + T_{CLAZ} (186 min)
= 0.5 (125) − 30 + 10 = 42.5 ns.

NOTE:
The hold time calculation is the same for both the 80186 and 8087.

These timings provide adequate setup and hold times for a 74LS373 address latch.

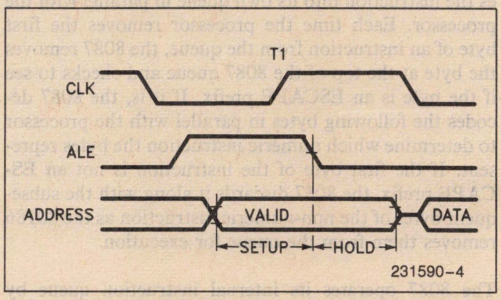
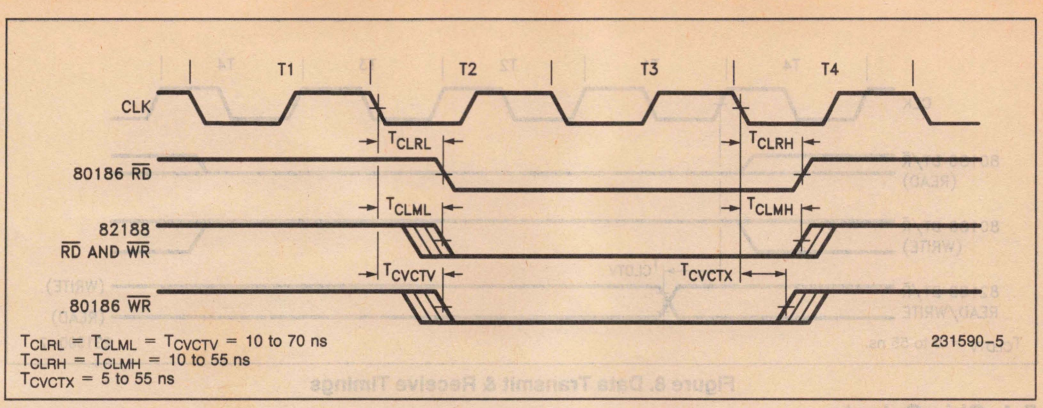


Figure 5. Address Latch Timings



$T_{CLRL} = T_{CLML} = T_{CVCTV} = 10$ to 70 ns
$T_{CLRH} = T_{CLMH} = 10$ to 55 ns
$T_{CVCTX} = 5$ to 55 ns

Figure 6. Read and Write Timings

5.3.2 Read and Write

The read and write signals of the 82188 have identical timings to those of the 80186 with one exception: the 82188 $\overline{WR}$ inactive edge may not go inactive quite as early as the 80186. This spec is, in fact, a tighter spec than the 80186 $\overline{WR}$ timing and should make designs easier. The timings for $\overline{RD}$ and $\overline{WR}$ are shown in Figure 6 for both the 80186 and the 82188.

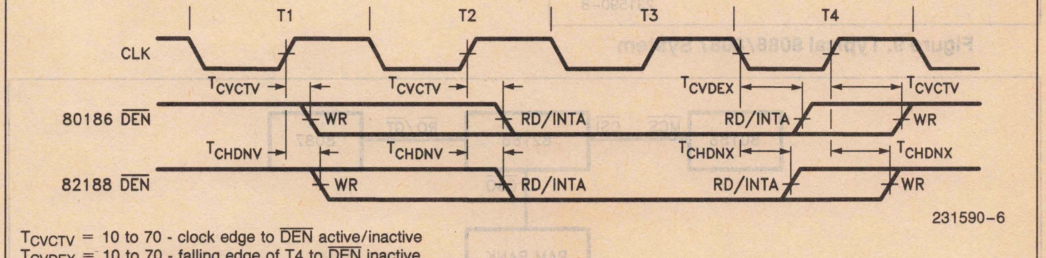
5.3.3 $\overline{DEN}$

The $\overline{DEN}$ signal on the 82188 is identical to the $\overline{DEN}$ signal on the 80186 but with a tighter timing specification. This makes designs easier with the 82188 and makes upgrades from 80186 bus control to 82188 bus control more straightforward. The timings for $\overline{DEN}$ on both the 80186 and 82188 are shown in Figure 7.

5.3.4 DT/$\overline{R}$

The operation of the DT/$\overline{R}$ signal varies somewhat between the 80186 and the 82188. The 80186 DT/$\overline{R}$ signal will remain in an active high state for all write cycles and will default to a high state when the system bus is idle (i.e., no bus activity). The 80186 DT/$\overline{R}$ goes low only for read cycles and does so only for the duration of the bus cycle. At the end of the read cycle, assuming the following cycle is a non-read, the DT/$\overline{R}$ signal will default back to a high state. Back-to-back read cycles will result in the DT/$\overline{R}$ signal remaining low until the end of the last read cycle.

The DT/$\overline{R}$ signal on the 82188 operates differently by making transitions only at the start of a bus cycle. The 82188 DT/$\overline{R}$ signal has no default state and therefore will remain in whichever state the previous bus cycle required. The 82188 DT/$\overline{R}$ signal will only change states when the current bus cycle requires a state different from the previous bus cycle.



T_{CVCTV} = 10 to 70 - clock edge to $\overline{DEN}$ active/inactive
T_{CVDEX} = 10 to 70 - falling edge of T4 to $\overline{DEN}$ inactive
T_{CHDNV} = 10 to 55 - rising edge of clock to $\overline{DEN}$ active
T_{CHDNX} = 10 to 55 - clock edge to $\overline{DEN}$ inactive

Figure 7. Data Control Timings

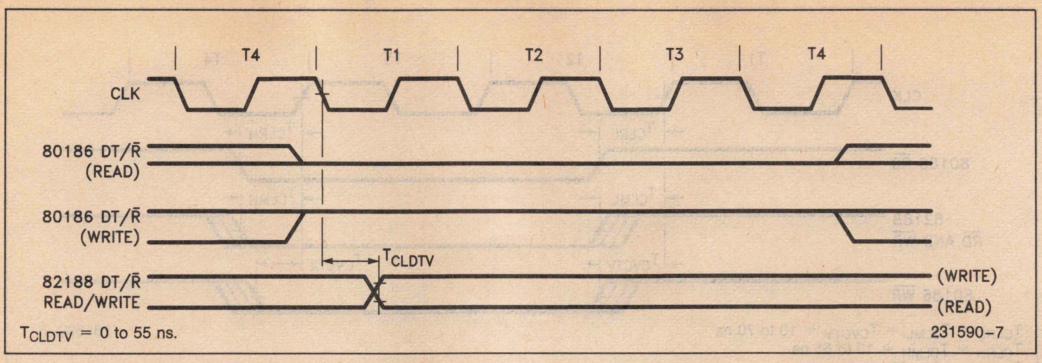


Figure 8. Data Transmit & Receive Timings

5.4 Chip Selects

5.4.1 INTRODUCTION

Chip-select circuitry is typically accomplished by using a discrete decoder to decode two or more of the upper address lines. When a valid address appears on the address bus, the decoder generates a valid chip-select. With this method, any bus master capable of placing an address on the system bus is able to generate a chip-select. An example of this is shown in Figure 9 where an 8086/8087 system uses a common decoder on the address bus. Note the decoder is able to operate regardless of which processor is in control of the bus.

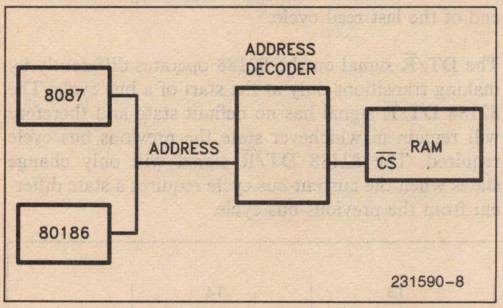


Figure 9. Typical 8086/8087 System

With high integration processors like the 80186 and 80188, the chip-select decoder is integrated onto the processor chip. The integrated chip-selects on the 80186 enable direct processor connection to the chip-enable pins on many memory devices, thus eliminating an external decoder. But because the integrated chip-selects decode the 80186's internal bus, an external bus master, such as the 8087, is unable to activate them. The 82188 IBC solves this problem by supplying a chip-select mechanism which may be activated by both the host processor and a second processor.

5.4.2 $\overline{CSI}$ AND $\overline{CSO}$ OF THE 82188

The $\overline{CSI}$ (chip select in) and $\overline{CSO}$ (chip select out) pins of the 82188 provide a way for a second bus master to select memory while also making use of the 80186 integrated chip-selects. The $\overline{CSI}$ pin of the 82188 connects directly to one of the 80186's chip-selects while $\overline{CSO}$ connects to the memory device designated for the chip-selects range. An example of this is shown in Figure 10.

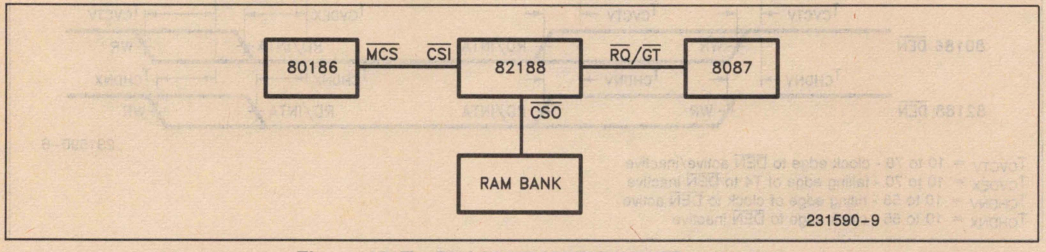


Figure 10. Typical 80186/82188/8087 System

When the 80186 has control of the bus, the circuit acts just as a buffer and the memory device gets selected as if the circuit had not been there. Whenever $\overline{CSI}$ goes active, $\overline{CSO}$ goes active. When a second bus master, such as the 8087, takes control of the bus, $\overline{CSO}$ goes active and remains active until the 8087 passes control back to the processor. At this time $\overline{CSO}$ is deactivated.

A functional block diagram of the $\overline{CSI}$–$\overline{CSO}$ circuit is shown in Figure 11. A grant pulse on the $\overline{RQ}/\overline{GT0}$ line gives control to the 8087 and also causes the $\overline{8087CONTROL}$ signal to go active, which in turn causes $\overline{CSO}$ to go active. The $\overline{8087CONTROL}$ signal goes inactive when either a release is received on $\overline{RQ}/\overline{GT0}$, indicating that the 8087 is relinquishing control to the main processor, or a grant is received on the $\overline{RQ}/\overline{GT1}$ line, indicating that the 8087 is relinquishing control to a third processor. Both actions signify that the 8087 is relinquishing the bus. If $\overline{CSO}$ goes inactive because a third processor took control of the bus, then $\overline{CSO}$ will go active again for the 8087 when a release pulse is transmitted on the $\overline{RQ}/\overline{GT1}$ line to the 8087. This release pulse occurs as a result of SYSHLDA going inactive from the third processor.

5.4.3 SYSTEM DESIGN EXAMPLE

To provide the 8087 access to data in low memory through an integrated chip-select, the $\overline{LCS}$ pin should be disconnected from the bank that it is currently selecting and fed directly into the 82188 $\overline{CSI}$. The $\overline{CSI}$ output should be connected to the banks which the $\overline{LCS}$ formerly selected. The $\overline{LCS}$ will still select the same banks because $\overline{CSO}$ goes active whenever $\overline{CSI}$ goes active. But now the 8087, when taking control of the bus, may also select these banks.

Care must be taken in locating the 8087 data area because it must reside in the area in which the chip-select is defined. If the 8087 generates an address outside of the $\overline{LCS}$ range, the $\overline{CSO}$ will still go active, but the address will erroneously select a part of the lower bank. Note also that this chip-select limits the size of the 8087 data area to the maximum size memory which can be selected with one chip-select. However, this does not place a limit on instruction code size or non-8087 data size. All 80186 and 8087 instructions are fetched by the processor and therefore do not require that the 8087 be

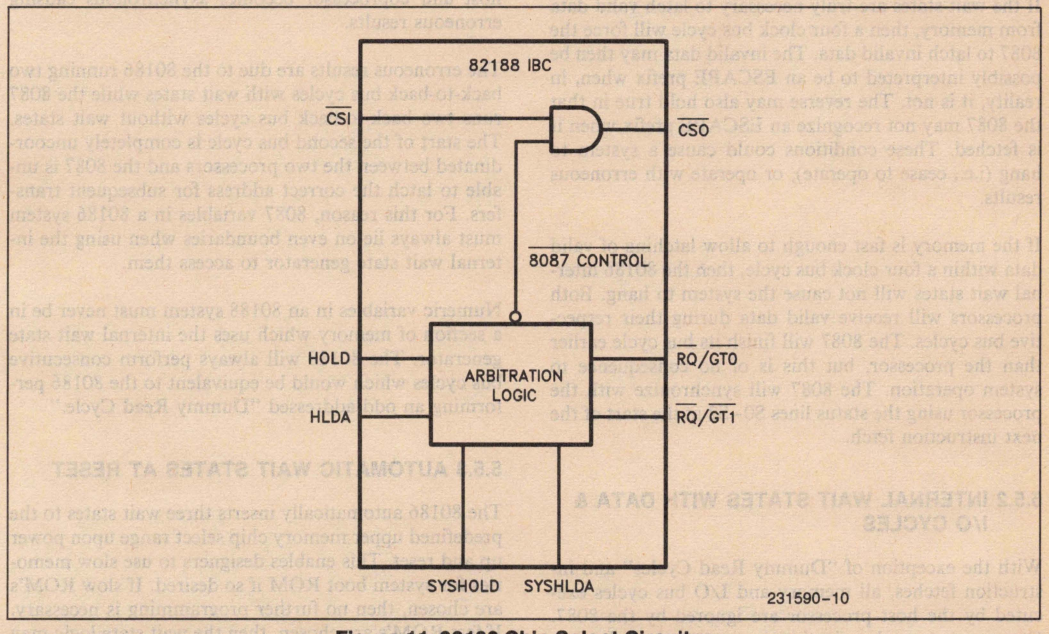


Figure 11. 82188 Chip Select Circuitry

able to address them. Likewise, non-8087 data is never accessed by the 8087 and therefore does not require an 8087 chip-select.

5.5 Wait State & Ready Logic

The 8087 must accurately track every instruction fetch the 80186 performs so that each op-code may be read from the system bus by the 8087 in parallel with the processor. This means that for instruction code areas, the 80186 cannot use internally generated wait states. All ready logic for these areas must be generated externally and sent into the 82188. The 82188 then presents a synchronous ready out (SRO) signal to both the 80186 and the 8087.

5.5.1 INTERNAL WAIT STATES WITH INSTRUCTION FETCHES

If internal wait states are used by the processor with the 8087 at zero wait states, then the 8087 will latch op-codes using a four clock bus cycle while the processor is using between five and seven clocks on each bus cycle. If the wait states are truly necessary to latch valid data from memory, then a four clock bus cycle will force the 8087 to latch invalid data. The invalid data may then be possibly interpreted to be an ESCAPE prefix when, in reality, it is not. The reverse may also hold true in that the 8087 may not recognize an ESCAPE prefix when it is fetched. These conditions could cause a system to hang (i.e., cease to operate), or operate with erroneous results.

If the memory is fast enough to allow latching of valid data within a four clock bus cycle, then the 80186 internal wait states will not cause the system to hang. Both processors will receive valid data during their respective bus cycles. The 8087 will finish its bus cycle earlier than the processor, but this is of no consequence to system operation. The 8087 will synchronize with the processor using the status lines S0–S2 at the start of the next instruction fetch.

5.5.2 INTERNAL WAIT STATES WITH DATA & I/O CYCLES

With the exception of "Dummy Read Cycles" and instruction fetches, all memory and I/O bus cycles executed by the host processor are ignored by the 8087. Coprocessor synchronization is not required for untracked bus cycles and, therefore, internally generated wait states do not affect system operation. All of the I/O space and any part of memory used strictly for data may use the internal wait state generator on the 80186.

Memory used for 8087 data is somewhat different. Here, as in the case of code segment areas, the system must rely on an external ready signal or else the memory must be fast enough to support zero wait state operation. Without an external ready signal, the 8087 will always perform a four clock bus cycle which, when used with slow memories, results in the latching of invalid data.

Internal wait states will not affect system operation for data cycles performed by the 8087. In this case the 8087 has control of the bus and the two processors operate independently.

One type of data cycle has not yet been considered. Each time a numerics variable is accessed, the host processor runs a "Dummy Read Cycle" in order to calculate the operand address for the 8087. The 8087 latches the address and then takes control of the bus to fetch any subsequent bytes which are necessary. If the 8087 variables are located at even addresses, then an internally generated wait state will not present any problems to the system. If any numeric variables are located at odd addresses, then the interface between the host and coprocessor becomes asynchronous causing erroneous results.

The erroneous results are due to the 80186 running two back-to-back bus cycles with wait states while the 8087 runs two back-to-back bus cycles without wait states. The start of the second bus cycle is completely uncoordinated between the two processors and the 8087 is unable to latch the correct address for subsequent transfers. For this reason, 8087 variables in a 80186 system must always lie on even boundaries when using the internal wait state generator to access them.

Numeric variables in an 80188 system must never be in a section of memory which uses the internal wait state generator. The 80188 will always perform consecutive bus cycles which would be equivalent to the 80186 performing an odd addressed "Dummy Read Cycle."

5.5.3 AUTOMATIC WAIT STATES AT RESET

The 80186 automatically inserts three wait states to the predefined upper memory chip select range upon power up and reset. This enables designers to use slow memories for system boot ROM if so desired. If slow ROM's are chosen, then no further programming is necessary. If fast ROM's are chosen, then the wait state logic may simply be reprogrammed to the appropriate number of wait states.

The automatic wait states have the possibility of presenting the same problem as described in section 5.5.1 if

the boot ROM needs one or more wait states. Under these conditions the 8087 would be forced to latch invalid opcodes and possibly mistake one for an ESCAPE instruction.

If the boot ROM requires wait states, then some sort of external ready logic is necessary. This allows both processors to run with the same number of wait states and insures that they always receive valid data.

If the boot ROM does not require wait states, then there is no need to design external ready logic for the upper chip select region. But if 8087 code is present in the upper memory chip select region, the situation described in section 3.4 regarding "Dummy Read Cycles" must be considered.

The 82188 solves this problem by inserting three wait states on the SRO line to the 8087 for the first 256 bus cycles. By doing this the 82188 inserts the same number of wait states to both processors keeping them synchronized. The initialization code for the 80186 must program the upper memory chip select to look at external ready and to insert zero wait states within these first 256 bus cycles. At the end of the 256 bus cycles, the 82188 stops inserting wait states and both processors run at zero wait states.

5.5.4 EXTERNAL READY SYNCHRONIZATION

The 80186 and 8087 sample READY on different clock edges. This implies that some sort of external synchronization is required to insure that both processors sample the same ready state. Without the synchronization, it would be possible for the external signal to change state between samples. The 80186 may sample ready high while the 8087 samples ready low. This would lead to the two processors running different length bus cycles and possibly cause the system to hang.

The 82188 provides ready synchronization through the ARDY and SRDY inputs. Once a valid transition is recorded, the 82188 presents the results on the SRO output and holds the output in that state until both processors have had a chance to sample the signal.

5.6 BUS ARBITRATION

In order for the 8087 to read and write numeric data to and from memory, it must have a means of taking control of the local bus. With the 8086/88 this is accomplished through a request-grant exchange protocol. The 80186, however, makes use of HOLD/HOLD ACKNOWLEDGE protocol to exchange control of the bus with another processor. The 82188 supplies the necessary conversion to interface $\overline{RQ}/\overline{GT}$ to HOLD/HLDA signals. The $\overline{RQ}/\overline{GT}$ signal of the 8087 connects directly to the 82188's $\overline{RQ}/\overline{GT}0$ input while the 82188's HOLD and HLDA pins connect to the 80186's HOLD and HLDA pins.

When the 8087 requires control of the bus, the 8087 sends a request on the $\overline{RQ}/\overline{GT}0$ line to the 82188. The 82188 responds by sending a HOLD request to the 80186. When HLDA is received back from the 80186, the 82188 sends a grant back to the 8087 on the same $\overline{RQ}/\overline{GT}0$ line.

The 82188 also has provisions for adding a third bus-master to the system which uses HOLD/HLDA protocol. This is accomplished by using the 82188 SYSHOLD, SYSHLDA, and $\overline{RQ}/\overline{GT}1$ signals. The third processor requests the bus by pulling the SYSHOLD line high. The 82188 will route (and translate if necessary) the requests to the current bus master. If the 8087 has control, the 82188 will request control via the $\overline{RQ}/\overline{GT}1$ line which should be connected to the 8087's $\overline{RQ}/\overline{GT}1$ line.

The 8087 will relinquish control by getting off the bus and sending a grant pulse on the $\overline{RQ}/\overline{GT}1$ line. The 82188 responds by sending a SYSHLDA to the third processor. The third processor lowers SYSHOLD when it has finished on the bus. The 82188 routes this in the form of a release pulse on the $\overline{RQ}/\overline{GT}1$ line to the 8087. The 8087 then continues bus activity where it left off. The maximum latency from SYSHOLD to SYSHLDA is equal to the 80186 latency + 8087 latency + 82188 latency.

5.7 SPEED REQUIREMENTS

One of the most important timing specs associated with the 80186-8087 interface is the speed at which the system should run. The 8087 was designed to operate with a 33% duty cycle clock whereas the 80186 and 80188 were designed to operate with a 50% duty cycle clock. In order to run both parts off the same clock, the 8087 must run at a slower speed than is typically implied by its dash number in the 8086/88 family.

To determine the speed at which an 8087 may run (with a 50% duty cycle clock), the minimum low and high times of the 8087 must be examined. The maximum of these two minimum specs becomes the half-period of the 50% duty cycle system clock. For example, the 8087-1 provides worst case spec compatibility with the 80186 at system clock-speeds of up to 8 MHz. The clock waveforms are shown in Figure 12 using 10 MHz timings.

The minimum clock low time spec (T_{CLCH}) of the 10 MHz 8087 is 53 ns. The clock low time of an 8 MHz 80186 is specified to be:

$$\frac{1}{2}(T_{CLCL}) - 7.5$$

Solving for T_{CLCL} of the 80186 using T_{CLCH} of the 8087 yields the following:

$$\frac{1}{2}(T_{CLCL}) - 7.5 = T_{CLCH}$$

$$(T_{CLCL}) = 2(T_{CLCH} + 7.5)$$

$$T_{CLCL} = 121 \text{ ns}$$

The calculation shows minimum cycle time of the 80186 to be 121 ns. This time translates into a maximum frequency of 8.26 MHz.

6.0 BENCHMARKS

6.1 Introduction

The following benchmarks compare the overall system performance of an 8086, 80188, and an 80186 in numeric applications. Results are shown for all three processors in systems with the 8087 coprocessor and in systems using an 8087 software emulator. Three FORTRAN benchmark programs are used to demonstrate the large increase in floating-point math performance provided by the 8087 and also the increase in performance due to the enhanced 80186 and 80188 host processors.

The 8086 results were measured on an Intellec® Series III Microcomputer Development System with an iSBC® 86/12 board and an iSBC 337 multimodule. Typically, one wait state for memory read cycles and two wait states for memory write cycles are experienced in this environment.

The 80186 and 80188 results were measured on a prototype board which allowed zero wait state operation at 8 MHz. The benchmarks measured using the 8087 showed little sensitivity to wait states. Instructions executed on the 8087 tend to be long in comparison to the amount of bus activity required and, therefore, are not affected much by wait states.

The benchmarks measured using the software emulator are much more bus intensive and average from 10 to 15 percent performance degradation for one wait state.

All execution times shown here represent 8 MHz operation. The 8086 results were measured at 5 MHz and extrapolated to achieve 8 MHz execution times.

6.2 Interest Rate Calculations

Routines were written in FORTRAN-86 to calculate the final value of a fund given the annual interest and the present value. It is assumed that the interest will be compounded daily, which requires the calculation of the yearly effective rate. This value, which is the equivalent annual interest if the interest were compounded daily, is determined by the following formula:

```
yer = (1 + (ir/np))**np - 1
```

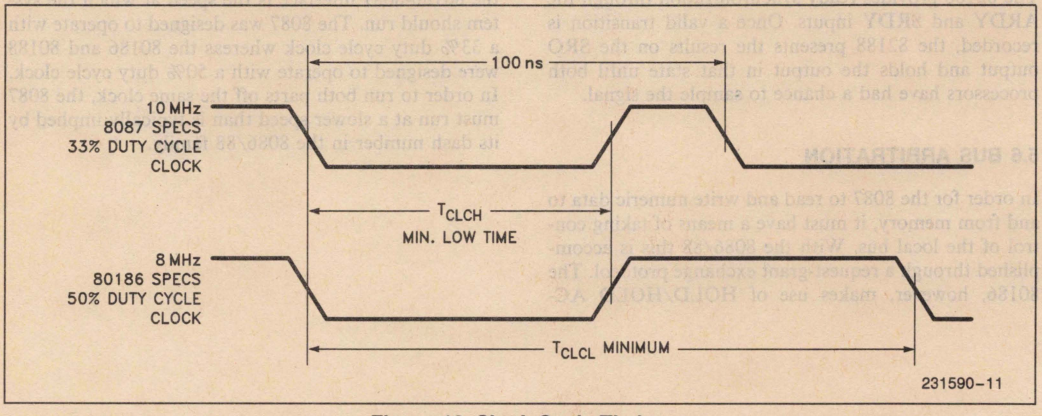


Figure 12. Clock Cycle Timing

where:

yer is the yearly effective rate
ir is the annual interest rate
np is the number of compounding periods per annum

Once the yer is determined, the final value of the fund is determined by the formula:

$$fv = (1 + yer) * pv$$

where:

pv is the present value
fv is the future value

Results are obtained using single-precision, double-precision, and temporary real precision operands when:

ir is set to 10% (0.1)
np is set to 365 (for daily compounding)
pv is set to $2,000,000

THE RESULTS:

	yer	Final Value
Single-Precision (32-bit)	10.514%	$2,210,287.50
Double-Precision (64-bit)	10.516%	$2,210,311.57
Temporary Real Precision	10.516%	$2,210,311.57

The difference between the final single-precision and double-precision values is $24.07; the difference in the final value between the double-precision and the temporary real precision is 0.000062 cents. Since the 8087 performs all internal calculations on 80-bit floating point numbers (temp real format), temporary real precision operations perform faster than single- or double-precision. No data conversions are required when loading or storing temporary real values in the 8087. Thus, for business applications, the double-precision computing of the 8087 is essential for accurate results, and the performance advantage of using the 8087 turns out to be as much as 100 times the equivalent software emulation program.

6.3 Matrix Multiply Benchmark Routine

A routine was written in FORTRAN-86 to compute the product of two matrices using a simple row/column inner-product method. Execution times were obtained for the multiplication of 32×32 matrices using double precision. The results of the benchmark are shown in Figure 14.

The results show the 8087 coprocessor systems performing from 23 to 31 times faster than the equivalent software emulation program. Both the 80188/87 and the 80186/87 systems outperform the 8086/87 system by 34 to 75 percent. This difference is mainly attributed to the fact that the matrix program largely consists of effective address calculations used in array accessing. The hardware effective address calculator of the 80186 and 80188 allow each array access to improve by as much as three times the 8086 effective address calculation.

6.4 Whetstone Benchmark Routine

The Whetstone benchmark program was developed by Harry Curnow for the Central Computer Agency of the British government. This benchmark has received high visibility in the scientific community as a measurement of main frame computer performance. It is a "synthetic" program. That is, it does not solve a real problem, but rather contains a mix of FORTRAN statements which reflect the frequency of such statements as measured in over 900 actual programs. The program computes a performance metric: "thousands of Whetstone instructions per second (KIPS)."

Simple variable and array addressing, fixed- and floating- point arithmetic, subroutine calls and parameter passing, and standard mathematical functions are performed in eleven separate modules or loops of a prescribed number of iterations.

Table 2. Interest Rate Benchmark Results

	8087 Software Emulator			8087 Coprocessor		
	80188	8086	80186	80188	8086	80186
Single Precision	70.3 ms	62.8 ms	43.4 ms	.70 ms	.66 ms	.61 ms
Double Precision	72.1 ms	62.9 ms	44.4 ms	.71 ms	.66 ms	.61 ms
Temp Real Precision	72.6 ms	63.0 ms	44.8 ms	.69 ms	.65 ms	.59 ms
Average	71.7 ms	62.9 ms	44.2 ms	.70 ms	.66 ms	.60 ms

The original coding of the Whetstone benchmark was written in Algol-60 and used single-precision values. It was rewritten in FORTRAN with single-precision values to exactly reflect the original intent. Another version was created using double-precision values. The results are shown in Table 3.

The results show the 8087 systems with the 80186 and 80188 outperforming the equivalent software emulation by 60 to 83 times. Additionally, the 80186 coupled with the 8087 outperformed the 8086/87 system by 22 percent.

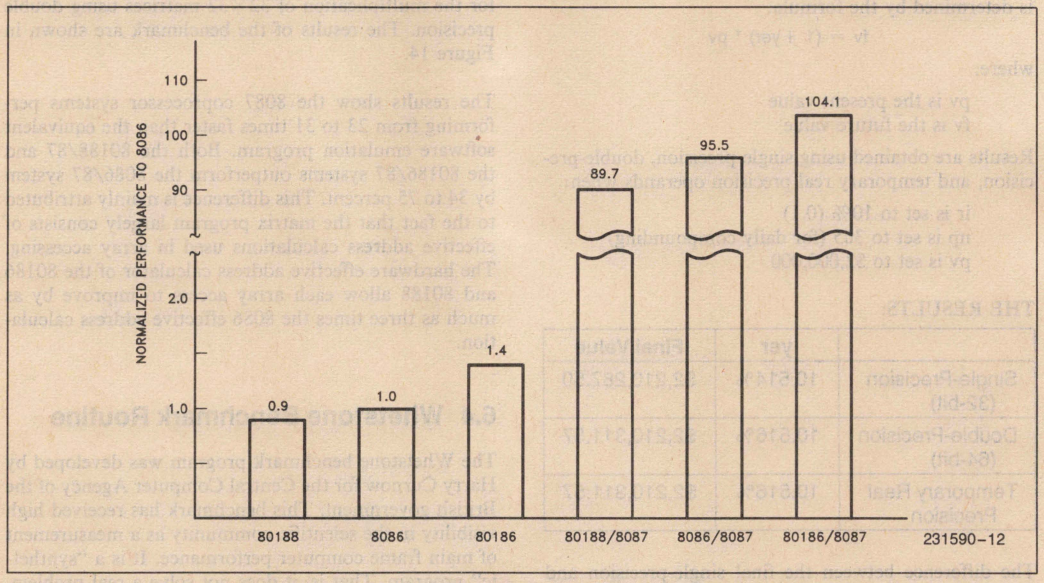


Figure 13. Interest Rate Benchmark Results

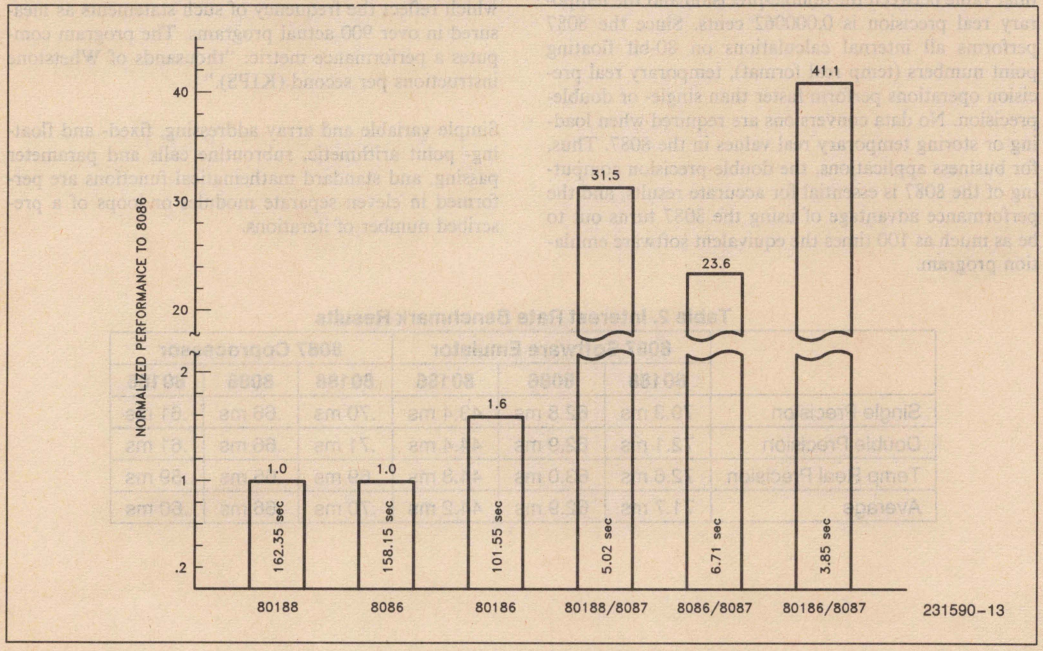


Figure 14. Double Precision Matrix Multiplication

Table 3. Whetstone Benchmark Results

Units = KIPS	8087 Software Emulator			8087 Coprocessor		
	80188	8086	80186	80188	8086	80186
Single Precision	2	2.3	3.3	165.8	178.0	197.6
Double Precision	2	2.2	3.2	151.7	152.0	185.2

6.5 Benchmark Conclusions

These benchmarks show that the 8087 Numeric Data Coprocessor, coupled with either the 80186 or the 80188, can increase the performance of a numeric application by 75 to 100 times the equivalent software emulation program.

Applications which require array accessing with effective address calculations will benefit even more by using the 80188 and 80186 as the host processor as compared to the 8086. The results of the matrix multiplication show both the 80188 and 80186 outperforming the 8086 by 34 and 75%, respectively, in an 8087 system. In general, an 80186/8087 system will offer a 10% to a 75% improvement over an equivalent 8086/8087 system, depending on the instruction mix.

7. CONCLUSION

For controller applications which require high performance in numerics and low system cost, the 16-bit 80186 or 8-bit 80188 coupled with the 8087 offers an ideal solution. The integrated features of the 80186 and 80188 offer a low system cost through reduced board space and a simplified production flow while the 8087 fulfills the performance requirements of numeric applications.

The 82188 IBC provides a straightforward, highly integrated solution to interfacing the 80188 or 80186 to the 8087. The bus control timings of the 82188 are compatible with the 80186 and 80188, allowing easy upgrades from existing designs. The 82188 features present a highly integrated solution to both new and old designs.

The coprocessing capabilities of the 8087 bring performance improvements of 75 to 100 times the equivalent 80186 or 80188 software emulation program and an 80186/8087 system will offer a 10% to a 75% improvement over an equivalent 8086/8087 system depending on the instruction mix.

In addition a growing base of high-level language support (FORTRAN, Pascal, C, Basic, PL/M, etc.) from Intel and numerous third-party software vendors facilitates the timely and efficient generation of application software.

REFERENCES:

82188 Data Sheet #231051
80186 Data Sheet #210451
80188 Data Sheet #210706
iAPX 86/88 80186/188 Users Manual
Programmers Reference #210911
Hardware Reference #210912
AP-113 "Getting Started with the
Numeric Data Processor #207865

intel

APPLICATION NOTE

AP-286

October 1986

80186/188 Interface to Intel Microcontrollers

PARVIZ KHODADADI
APPLICATIONS ENGINEER

Order Number: 231784-001

1.0 INTRODUCTION

Systems which require I/O processing and serial data transmission are very software intensive. The communication task and I/O operations consume a lot of the system's intelligence and software. In many conventional systems the central processing unit carries the burden of all the communication and I/O operations in addition to its main routines, resulting in a slow and inefficient system.

In an ideal system, tasks are divided among processors to increase performance and achieve flexibility. One attractive solution is the combination of the Intel highly integrated 80186 microprocessor and the Intel 8-bit microcontrollers such as the 80C51, 8052, or 8044. In such a system, the 80186 provides the processing power and the 1 Mbyte memory addressability, while the controller provides the intelligence for the I/O operations and data communication tasks. The 80186 runs application programs, performs the high level communication tasks, and provides the human interface. The microcontroller performs 8-bit math and single bit boolean operations, the low level communication tasks, and I/O processing.

This application note describes an efficient method of interfacing the 16-bit 80186 high integration microprocessor to the 80C51, 8052, or the microcontroller-based 8044 serial communication controller. The interface hardware shown in Figure 1.1, is very simple and may be implemented with a programmable logic device or a gate-array. The 80186 and the microcontroller may run asynchronously and at different speeds. With this technique data transfers up to 200 Kbytes per second can be achieved between a 12 MHz microcontroller and an 8 MHz 80186.

The 8-bit 80188 high integration microprocessor can also be used with the same interface technique. The performance of the interface is the same since an 8-bit bus is used.

Interface to the 8044, 80C51, and the 8052 is identical because they have identical pinouts (some pins have alternate functions). As an example, the software procedures for the 8044/80186 interface, which is the building block for the application driver, is supplied in this Application Note.

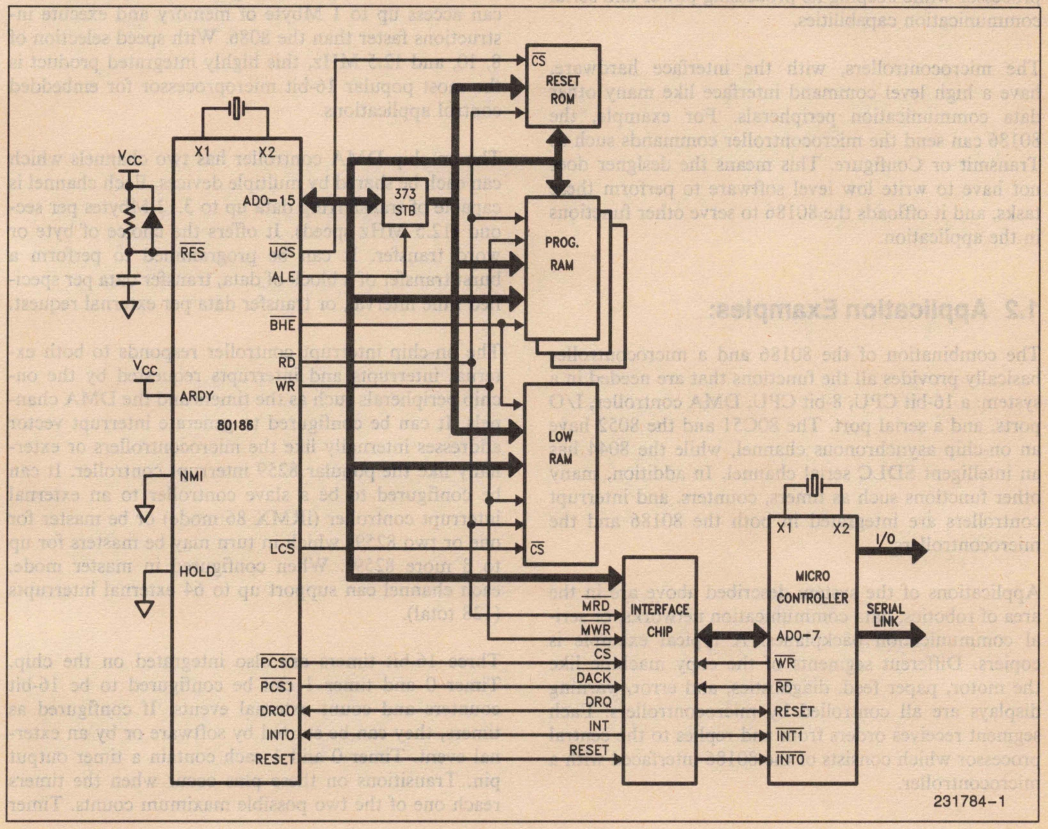


Figure 1.1. 80186/Microcontroller Based System

1.1 System Overview

The 80186 and the microcontrollers are processors. They each access memory and have address/data, read, and write signals. There are three common ways to interface multiple processors together:

1) First In First Out (FIFO)

2) Dual Port RAM (DPRAM)

3) Slave Port

The FIFO interface, compared to DPRAM, requires less TTL and is easier to interface; however, FIFOs are expensive. The DPRAM interface is also expensive and even more complex. When DPRAM is used, the address/data lines of each processor must be buffered, and hardware logic is needed to arbitrate access to DPRAM. The slave port interface given here is cheaper and easier than both FIFO and DPRAM alternatives.

The 80186 processor, when interfaced to this circuit, views the microcontroller as a peripheral chip with 8-bit data bus and no address lines (see Figure 1.1). It can read status and send commands to the microcontroller at any time. The microcontroller becomes a slave co-processor while keeping its processing power and serial communication capabilities.

The microcontrollers, with the interface hardware, have a high level command interface like many other data communication peripherals. For example, the 80186 can send the microcontroller commands such as Transmit or Configure. This means the designer does not have to write low level software to perform these tasks, and it offloads the 80186 to serve other functions in the application.

1.2 Application Examples:

The combination of the 80186 and a microcontroller basically provides all the functions that are needed in a system: a 16-bit CPU, 8-bit CPU, DMA controller, I/O ports, and a serial port. The 80C51 and the 8052 have an on-chip asynchronous channel, while the 8044 has an intelligent SDLC serial channel. In addition, many other functions such as timers, counters, and interrupt controllers are integrated in both the 80186 and the microcontrollers.

Applications of the system described above are in the area of robotics, data communication networks, or serial communication backplanes. A typical example is copiers. Different segments of the copy machine like the motor, paper feed, diagnostics, and error/warning displays are all controlled by microcontrollers. Each segment receives orders from and replies to the central processor which consists of the 80186 interfaced with a microcontroller.

Another common application is in the area of process controllers. An example is a central control unit for a multiple story building which controls the heating, cooling, and lighting of each room in each floor. In each room a microcontroller performs the above functions based on the orders received from the central processor. Depending on the throughput and type of the serial communication required, the 8044 or the 80C51 (8052) may be selected for the application.

2.0 OVERVIEW OF THE 80186, 80C51, 8052, AND 8044

This section briefly discusses the features of the microcontrollers and the 80186. For more information about these products please refer to the Intel Microcontroller and Microsystem components hand-books. Readers familiar with the above products may skip this section.

2.1 The 80186 Internal Architecture

The 80186 contains an enhanced version of Intel's popular 8086 CPU integrated with many other features common to most systems (Figure 2.1). The 16-bit CPU can access up to 1 Mbyte of memory and execute instructions faster than the 8086. With speed selection of 8, 10, and 12.5 MHz, this highly integrated product is the most popular 16-bit microprocessor for embedded control applications.

The on-chip DMA controller has two channels which can each be shared by multiple devices. Each channel is capable of transferring data up to 3.12 Mbytes per second (12.5 MHz speed). It offers the choice of byte or word transfer. It can be programmed to perform a burst transfer of a block of data, transfer data per specified time interval, or transfer data per external request.

The on-chip interrupt controller responds to both external interrupts and interrupts requested by the on-chip peripherals such as the timers and the DMA channels. It can be configured to generate interrupt vector addresses internally like the microcontrollers or externally like the popular 8259 interrupt controller. It can be configured to be a slave controller to an external interrupt controller (iRMX 86 mode) or be master for one or two 8259s which in turn may be masters for up to 8 more 8259s. When configured in master mode, each channel can support up to 64 external interrupts (128 total).

Three 16-bit timers are also integrated on the chip. Timer 0 and timer 1 can be configured to be 16-bit counters and count external events. If configured as timers, they can be started by software or by an external event. Timer 0 and 1 each contain a timer output pin. Transitions on these pins occur when the timers reach one of the two possible maximum counts. Timer

2 can be used as a prescaler for timer 0 and 1 or can be used to generate DMA requests to the on-chip DMA channel.

Finally, the integrated clock generator, the wait state generator, and the chip select logic reduce the external logic necessary to build a processing system.

2.2 The MCS-51 Internal Architecture

The 80C51BH, as shown in Figure 2.2, consists of an 8-bit CPU which can access up to 64 Kbytes of data memory (RAM) and 64 Kbytes of program memory (ROM). In addition, 4 Kbytes of ROM and 128 bytes of RAM are built onto the chip.

The on-chip interrupt controller supports five interrupts with two priority levels. There are two timers integrated in the 80C51. Timer 0 and 1 can be configured as 8-bit or 16-bit timers or event counters.

Finally the integrated full duplex asynchronous serial channel provides the human interface or communication capability with other microcontrollers. The UART supports data rates up to 500 kHz (with 15 MHz crystal) and can distinguish between address bytes and data bytes.

The 8052 has the same features as the 80C51 except it has 8 Kbytes of on-chip ROM and 256 bytes of on-chip RAM. In addition the 8052 has another timer which may be configured as the baud rate generator for the serial port.

2.3 The 8044 Internal Architecture

The 8044 has all the features of the 80C51. In addition the on-chip RAM size is increased to 192 bytes and an intelligent HDLC/SDLC serial channel (SIU) replaces the 80C51 serial port (see Figure 2.3). It supports data rates up to 2.4 Mbps when an external clock is used and 375 Kbps when the clock is extracted from the data line. The serial port can be used in half duplex point to point, multipoint, or one-way loop configurations.

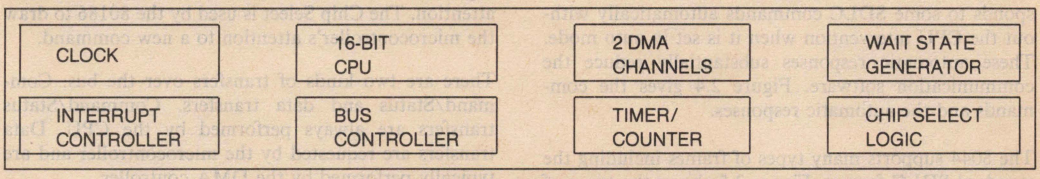


Figure 2.1. 80186 Block Diagram

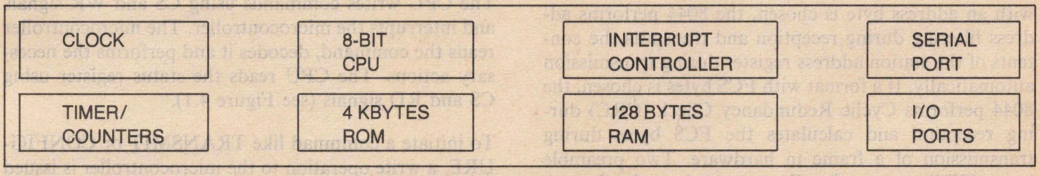


Figure 2.2. 80C51 Block Diagram

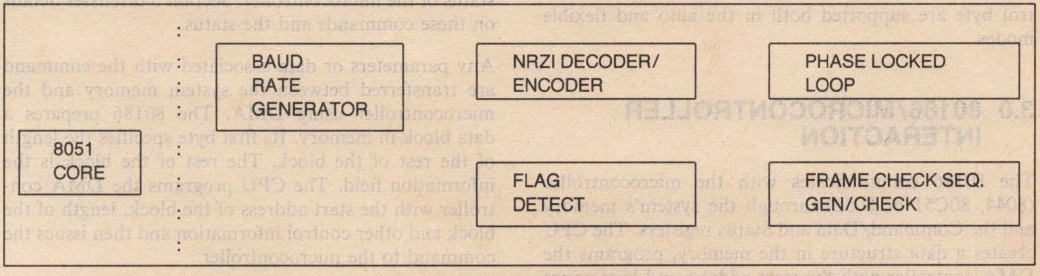


Figure 2.3. 8044 Block Diagram

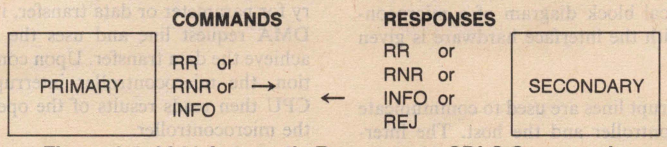


Figure 2.4. 8044 Automatic Response to SDLC Commands

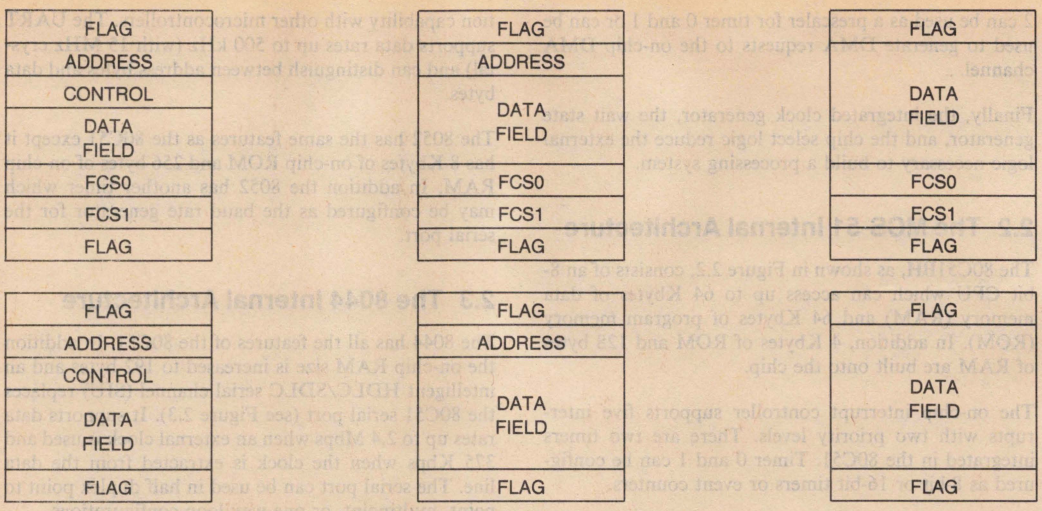


Figure 2.5. The 8044 Frame Formats

The SIU is called an intelligent channel because it responds to some SDLC commands automatically without the CPU intervention when it is set in auto mode. These automatic responses substantially reduce the communication software. Figure 2.4 gives the commands and the automatic responses.

The 8044 supports many types of frames including the standard SDLC format. Figure 2.5 shows the types of frames the 8044 can transmit and receive. If a format with an address byte is chosen, the 8044 performs address filtering during reception and transmits the contents of the station address register during transmission automatically. If a format with FCS bytes is chosen, the 8044 performs Cyclic Redundancy Check (CRC) during reception and calculates the FCS bytes during transmission of a frame in hardware. Two preamble bytes (PFS) may optionally be added to the frames. Formats that include the station address and the control byte are supported both in the auto and flexible modes.

3.0 80186/MICROCONTROLLER INTERACTION

The 80186 communicates with the microcontroller (8044, 80C51 or 8052) through the system's memory and the Command/Data and Status registers. The CPU creates a data structure in the memory, programs the DMA controller with the start address and byte count of the block, and issues a command to the microcontroller. A hypothetical block diagram of a microcontroller when used with the interface hardware is given in Figure 3.1.

Chip select and interrupt lines are used to communicate between the microcontroller and the host. The interrupt is used by the microcontroller to draw the 80186's attention. The Chip Select is used by the 80186 to draw the microcontroller's attention to a new command.

There are two kinds of transfers over the bus: Command/Status and data transfers. Command/Status transfers are always performed by the CPU. Data transfers are requested by the microcontroller and are typically performed by the DMA controller.

The CPU writes commands using CS and WR signals and interrupts the microcontroller. The microcontroller reads the command, decodes it and performs the necessary actions. The CPU reads the status register using CS and RD signals (see Figure 4.1).

To initiate a commnad like TRANSMIT or CONFIGURE, a write operation to the microcontroller is issued by the CPU. A read operation from the CPU gives the status of the microcontroller. Section 5 discusses details on these commands and the status.

Any parameters or data associated with the command are transferred between the system memory and the microcontroller using DMA. The 80186 prepares a data block in memory. Its first byte specifies the length of the rest of the block. The rest of the block is the information field. The CPU programs the DMA controller with the start address of the block, length of the block and other control information and then issues the command to the microcontroller.

When the microcontroller requires access to the memory for parameter or data transfer, it activates the 80186 DMA request line and uses the DMA controller to achieve the data transfer. Upon completion of an operation, the microcontroller interrupts the 80186. The CPU then reads results of the operation and status of the microcontroller.

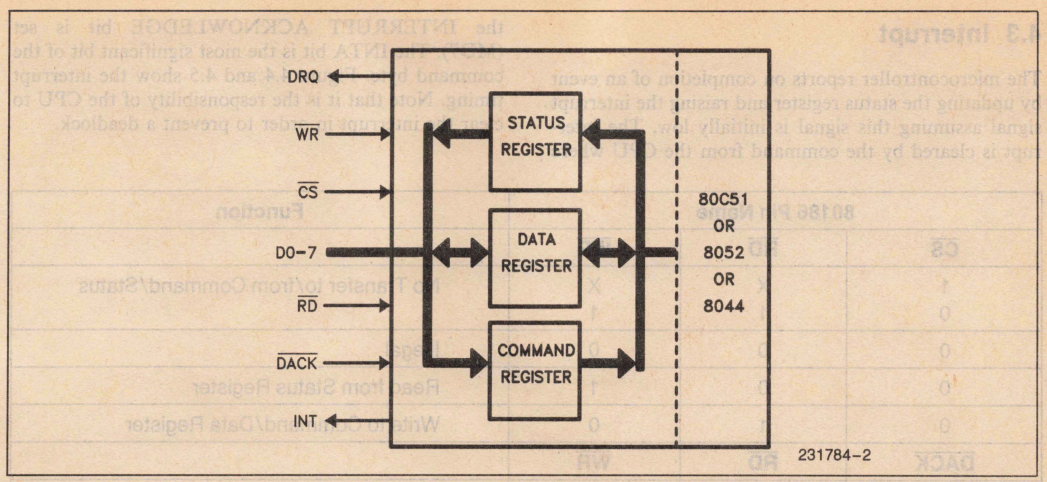


Figure 3.1. Microcontroller Plus the Interface Hardware Block Diagram

4.0 SYSTEM INTERFACE

There are two kinds of transfers over the bus: command/status and data transfers. The command/status transfers are always initiated and performed by the 80186. The data transfers are requested by the microcontroller using the DMA request (DRQ) line. In relatively slow systems the 80186 might also perform the data transfers. In that case, the request from the microcontroller will serve as an interrupt to the CPU. This mode of operation depends on the serial data rate.

The system interface performs command/status transfers, data/parameter transfers, and interrupts. This section describes the interface between the 80186 and a microcontroller shown in Figure 1.1. Section 6 describes the interface hardware.

4.1 Command/Status Transfers

The 80186 controls the microcontroller by writing into the command/data register and reading from the status register. The CPU writes a command by activating the chip select (PCS0), putting the command onto the data bus, and activating the WR signal. The command byte is latched into the command/data register, and the microcontroller is interrupted. In the interrupt service routine, the microcontroller reads the command byte from the command/data register, decodes the command byte, and activates the DRQ for data or parameter transfer if the decoded command requires such transfer.

At the end of parameter transfer the microcontroller updates the status register and interrupts the 80186.

4.2 Data/Parameter Transfer

Data/parameter transfers are controlled by a pair of REQUEST/ACKNOWLEDGE lines: DMA Request line (DRQ) and DMA Acknowledge line (DACK). Data and parameters are transferred via the Command/Data register to or from memory.

In order to request a transfer from memory, the microcontroller activates the DRQ pin. The DRQ signal goes active after a read operation by the microcontroller. In response, the 80186 DMA controller performs a byte transfer from the memory to the Command/Data register. Data is transferred on the bus and written into the Command/Data register on the rising edge of the 80186 WR signal ($\overline{MWR}$), which is activated by the DMA controller. Figure 4.2 shows the write timing.

In order to request a transfer to memory, the microcontroller activates the DRQ signal and outputs the data into the Command/Data latch. When the microcontroller WR signal goes active, DRQ is set. In response, the DMA performs the data transfer and resets the DRQ signal. Figure 4.3 shows the read timing.

4.3 Interrupt

The microcontroller reports on completion of an event by updating the status register and raising the interrupt signal assuming this signal is initially low. The interrupt is cleared by the command from the CPU where the INTERRUPT ACKNOWLEDGE bit is set (MD7). The INTA bit is the most significant bit of the command byte. Figure 4.4 and 4.5 show the interrupt timing. Note that it is the responsibility of the CPU to clear the interrupt in order to prevent a deadlock.

80186 Pin Name			Function
$\overline{CS}$	$\overline{RD}$	$\overline{WR}$	
1 0	X 1	X 1	No Transfer to/from Command/Status
0	0	0	Illegal
0	0	1	Read from Status Register
0	1	0	Write to Command/Data Register
$\overline{DACK}$	$\overline{RD}$	$\overline{WR}$	
1 0	X 1	X 1	No Transfer
0	0	0	Illegal
0	0	1	Data Read from DMA Channel
0	1	0	Data Write to DMA Channel

NOTE:
Only one of CS, DACK may be active at any time.

Figure 4.1. Data Bus Control Signals and Their Functions

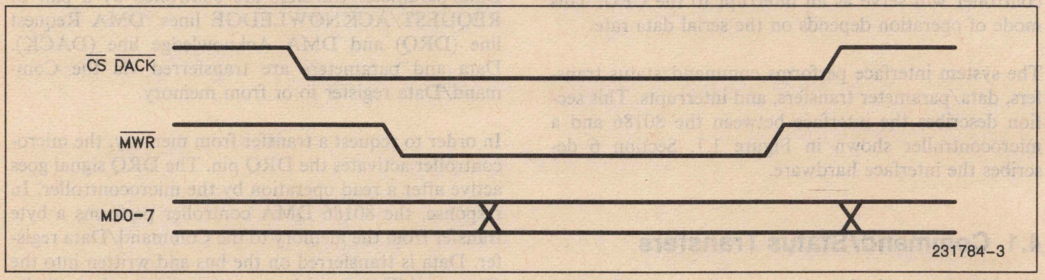


Figure 4.2. Write Timing

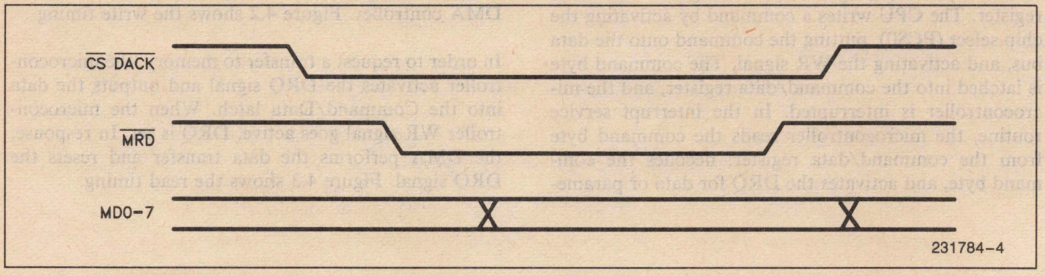


Figure 4.3. Read Timing

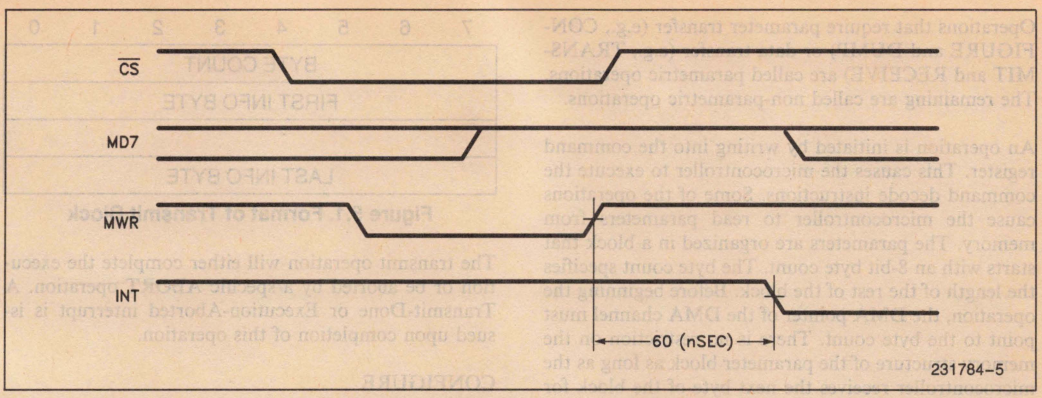


Figure 4.4. Interrupt Timing (Going Inactive)

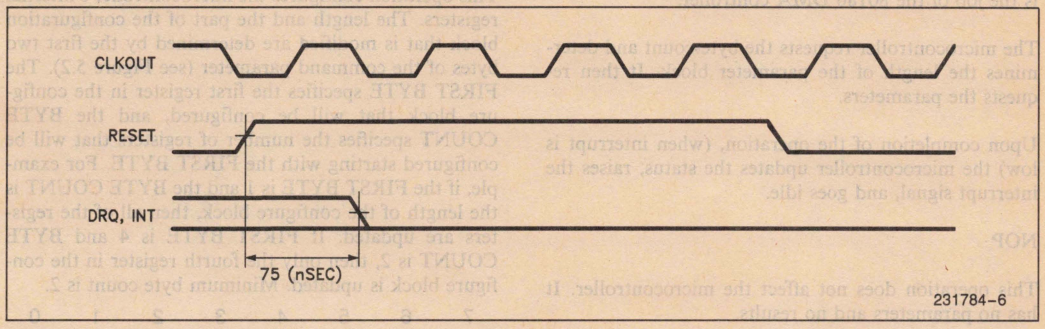


Figure 4.5. Reset Timing

5.0 COMMANDS AND STATUS

This section specifies the format of the commands and status. The commands and status given here are similar to most common coprocessors and data communication peripherals (e.g., the 82588 and 82586). The user may add more commands or redefine the formats for his/her own specific application.

5.1 Commands

A command is given to the microcontroller by writing it into the Command/Data register and interrupting the microcontroller. The command can be issued at any time; but in case it is not accepted, the operation is treated like a NOP and will be ignored (although the INT will be updated).

Format:

7	6	5	4	3	2	1	0
INTA	X	X	X	OPERATION			

5.1.1 ACKNOWLEDGING INTERRUPT (BIT 7)

The INTA bit, if set, causes the interrupt hardware signal and the interrupt bit to be cleared. This is the only way to clear the interrupt bit and reset the 80186 interrupt signal other than by a hardware reset.

5.1.2 OPERATIONS (BITS 0–3)

The OPERATION field initiates a specific operation. The microcontroller executes the following commands in software:

NOP
ABORT
TRANSMIT*
CONFIGURE*
DUMP*
RECEIVE*
TRA-DISABLE
REC-DISABLE
*Requires DMA operation.

The above operations except ABORT are executed only when the microcontroller is not executing any other operation. Abort is accepted only when the CPU is performing a DMA operation.

Operations that require parameter transfer (e.g., CONFIGURE and DUMP) or data transfer (e.g., TRANSMIT and RECEIVE) are called parametric operations. The remaining are called non-parametric operations.

An operation is initiated by writing into the command register. This causes the microcontroller to execute the command decode instructions. Some of the operations cause the microcontroller to read parameters from memory. The parameters are organized in a block that starts with an 8-bit byte count. The byte count specifies the length of the rest of the block. Before beginning the operation, the DMA pointer of the DMA channel must point to the byte count. There is no restriction on the memory structure of the parameter block as long as the microcontroller receives the next byte of the block for every DMA request it generates. Transferring the bytes is the job of the 80186 DMA controller.

The microcontroller requests the byte-count and determines the length of the parameter block. It then requests the parameters.

Upon completion of the operation, (when interrupt is low) the microcontroller updates the status, raises the interrupt signal, and goes idle.

NOP

This operation does not affect the microcontroller. It has no parameters and no results.

ABORT

This operation attempts to abort the completion of an operation under execution. It is valid for CONFIGURE, TRANSMIT, DUMP, and RECEIVE. It is ignored for any of the above if transfer of parameters has already been accomplished. The microcontroller, upon reception of the ABORT command, stops the DMA operation and issues an Execution-Aborted interrupt.

TRANSMIT

This operation transmits one message. A message may be transmitted as an SDLC frame by the 8044, or in ASYNC protocol by the 80C51 or the 8052 serial port.

Figure 5.1 shows the format of the Transmit block. A typical transmit operation parameter block includes the destination address and the control byte in the information field. As an example, see the 8044 transmit block in Figure 7.2.

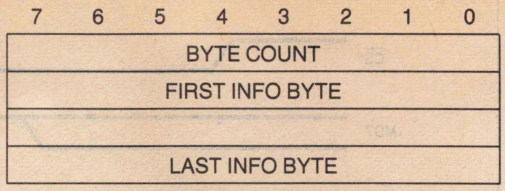


Figure 5.1. Format of Transmit Block

The transmit operation will either complete the execution or be aborted by a specific ABORT operation. A Transmit-Done or Execution-Aborted interrupt is issued upon completion of this operation.

CONFIGURE

This operation configures the microcontroller's internal registers. The length and the part of the configuration block that is modified are determined by the first two bytes of the command parameter (see Figure 5.2). The FIRST BYTE specifies the first register in the configure block that will be configured, and the BYTE COUNT specifies the number of registers that will be configured starting with the FIRST BYTE. For example, if the FIRST BYTE is 1 and the BYTE COUNT is the length of the configure block, then all of the registers are updated. If FIRST BYTE is 4 and BYTE COUNT is 2, then only the fourth register in the configure block is updated. Minimum byte count is 2.

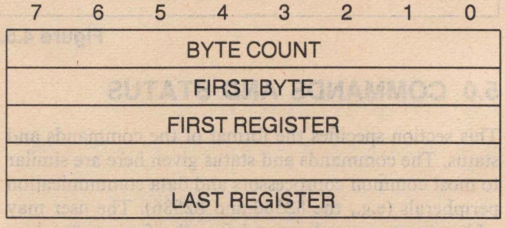


Figure 5.2. Format of Configure Block

A Configure-Done interrupt is issued when the operation is done unless ABORT was issued during the DMA operation.

DUMP

This operation causes dumping of a set of microcontroller internal registers to system memory. Figure 7.4 shows the format of the 8044 DUMP block.

The DUMP operation will either complete the execution or be aborted by a specific ABORT operation. A Dump-Done or Execution-Aborted interrupt is issued upon completion of this operation.

RECEIVE

This operation enables the reception of frames. It is ignored if the microcontroller's serial channel is already in reception mode.

The serial port receives only frames that pass the address filtering. The microcontroller transfers the received information and the byte count to the system memory using DMA. The completion of frame reception causes a Receive-Done event.

REC-DISABLE

This operation causes reception to be disabled. If transfer of data to the 80186 memory has already begun, then it is treated like the ABORT command. This operation has no parameters. REC-DISABLE is accepted only when the microcontroller's serial port is in receive mode.

TRA-DISABLE

This operation causes the transmission process to be aborted. If the microcontroller is fetching data from 80186 memory, then it is treated like the ABORT command. This operation has no parameters. It is accepted only when the serial port is in transmit mode.

5.1.3 ILLEGAL COMMANDS

Parametric and non-parametric commands except ABORT will be rejected (interrupt will not be set) if the microcontroller is already executing a command.

ABORT is rejected if issued when the microcontroller is not requesting DMA operation, or a non-Parametric execution is performed, or transfer of parameters/data has already been accomplished.

DMA operations shall not be aborted by any non-parametric or parametric command except by the ABORT command.

REC-DISABLE and TRA-DISABLE will not be accepted if the serial channel is idle.

5.2 Status

The microcontroller provides the information about the last operation that was executed, via the status register.

The microcontroller reports on these events by updating a status register and raising the INTERRUPT signal. Information from the status register is valid provided the interrupt signal is high or bit 0 of the status being read is set.

Format:

7	6	5	4	3	2	1	0
CTS*	RTS*	E	EVENT			DMA	INT

*8044 only

5.2.1 INTERRUPT (BIT 0)

The interrupt bit is set together with the hardware interrupt signal. Setting the INT bit indicates the occurrence of an event. This bit is cleared by any command whose INTA bit is set. Status is valid only when this bit is set.

5.2.2 DMA OPERATION (BIT 1)

The DMA bit, when set, indicates that a DMA operation is in progress. This bit is set if the commnad received by the microcontroller requires data or parameter transfer. If this bit is clear, DRQ will be inactive. The DMA bit, when cleared, indicates the completion of a DMA operation.

5.2.3 ERROR (BIT 5)

The E bit, if set, indicates that the event generated for the operation that was completed contains a warning, or the operation was not accepted.

5.2.4 REQUEST TO SEND (BIT 6)

The RTS bit, if clear, indicates that the serial channel is requesting a transmission.

5.2.5 CLEAR TO SEND (BIT 7)

The CTS bit indicates that, if the RTS bit is clear, the serial port is active and transmitting a frame.

5.2.6 EVENT (BITS 2-4)

The event field specifies why the microcontroller needs the attention of the 80186.

The following events may occur:

CONFIGURE-DONE
TRANSMIT-DONE
DUMP-DONE
RECEIVE-DONE
RECEPTION-DISABLED
TRANSMISSION-DISABLED
EXECUTION-ABORTED

CONFIGURE-DONE

This event indicates the completion of a CONFIGURE operation.

TRANSMIT-DONE

This event indicates the completion of the TRANSMIT operation.

If the E bit is set, it indicates that the transmit buffer was already full.

DUMP-DONE

This event indicates that the DUMP operation is completed.

RECEIVE-DONE

This event indicates that a frame has been received and stored in memory.

The format of the received message is indicated in Figure 5.3.

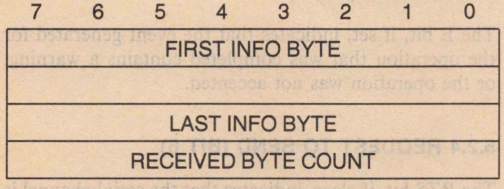


Figure 5.3. Format of Receive Block

Following the byte count, a few more bytes relating to the received frame such as the source address and the control byte may be transferred to the system memory using DMA. As an example, see the 8044 receive block in Figure 7.3.

Note that the format of a frame received by the microcontroller serial channel is configured by the CONFIGURE command.

If the E bit is set, buffer overrun has occurred.

RECEPTION-DISABLED

This event is issued as a result of a RCV-DISABLE operation that causes part of a frame to be disabled.

If the E bit is set, the serial port was already disabled, and the RCV-DISABLE is not accepted.

TRANSMISSION-DISABLED

This event is issued as a result of a TRA-DISABLE operation that causes transmission of a frame to be disabled.

The E bit, if set, indicates that the TRA-DISABLE operation was not accepted since the serial port was already idle, or transmission of a frame has already been accomplished.

EXECUTION-ABORTED

This event indicates that the execution of the last operation was aborted by the ABORT command.

If the E bit is set, ABORT was issued when the microcontroller was not executing any commands.

6.0 HARDWARE DESCRIPTION

The interface hardware shown in Figures 6.1 and 6.2 are identical. The difference is the status register. In Figure 6.2, an external latch is used to latch the status byte. This hardware is recommended if an extra I/O port on the microcontroller is required for some other applications, or external program and data memory is required for the microcontroller. The hardware shown in Figure 6.1 makes use of one of the microcontroller's I/O ports (Port 1) to latch the status to minimize hardware. The discussion of Sections 1 through 5 apply to both schematics.

6.1 Reset

After an 80186 hardware reset, the microcontroller is also reset. The on-chip registers are initialized as explained in the Intel Microcontroller Handbook. The reset signal also clears the 80186 interrupt and the microcontroller interrupt signals by resetting FF3 (Flip-Flop 3) and FF2 (Flip-Flop 2). Figure 4.5 shows the RESET timing.

6.2 Sending Commands

A bidirectional latched transceiver (74ALS646) is used for the Command/Data register. When the 80186 writes a command to the Command/Data register, it interrupts the microcontroller. The interrupt is generated only when bit 7 (INTA) of the command byte is set. When the 80186 PCS0 and WR signals go active to write the command, FF2 will be set and FF3 will be cleared. The output of FF3 is the interrupt to the 80186 and the INT status bit. The INT bit is cleared immediately to indicate that the status is no longer valid. The output of FF2 is the interrupt to the microcontroller. A high to low transition on this line will interrupt the microcontroller. The interrupt signal will be cleared as soon as the microcontroller reads the command from the Command/Data register.

6.3 DMA Transfers

In the interrupt service routine the command is decoded. If it requires a DMA transfer, the microcontroller sets the DMA bit of the status register which activates the DMA request signal. DRQ active causes the 80186 on-chip DMA to perform a fetch and a deposit bus cycle. The first DMA cycle clears the DRQ signal (FF1 is cleared). When the microcontroller performs a read or write operation, the output of the FF1 will be set, and DRQ goes active again.

The DMA controller transfers a byte from system memory to the Command/Data register. Data is latched when the 80186 PCS1 and WR signals go active. PCS1 and WR active also clear FF1. The microcontroller monitors the output of FF1 by polling the P3.3 pin. When FF1 is cleared the microcontroller reads the byte from the Command/Data register. The P3.3 pin is also the interrupt pin. If a slow rate of transfer is acceptable, every DMA transfer can be interrupt driven to allow the microcontroller to perform other tasks.

The DMA controller transfers a byte from the Command/Data register to system memory by activating the 80186 PCS1 and RD signals. PCS1 and RD active also clear FF1. When FF1 is cleared the microcontroller writes the next byte to the Command/Data register.

When all the data is transferred, the microcontroller clears the DMA status bit to disable DRQ. It then updates the status, sets the INT bit, and interrupts the 80186.

If the interface hardware in Figure 6.1 is used P1.1 is the DMA status bit and P1.0 is the INT bit. The microcontroller enables or disables them by writing to port 1. In Figure 6.2, DRQ or INT is disabled or enabled by writing to the 74LS374 status register. Note that the INT status bit is cleared by the hardware when the 80186 writes a command.

6.4 Reading Status

The command is written and the status is read with the same chip select (PCS0), although the status is read through the 74LS245 transceiver and the command is written to the Command/Data register.

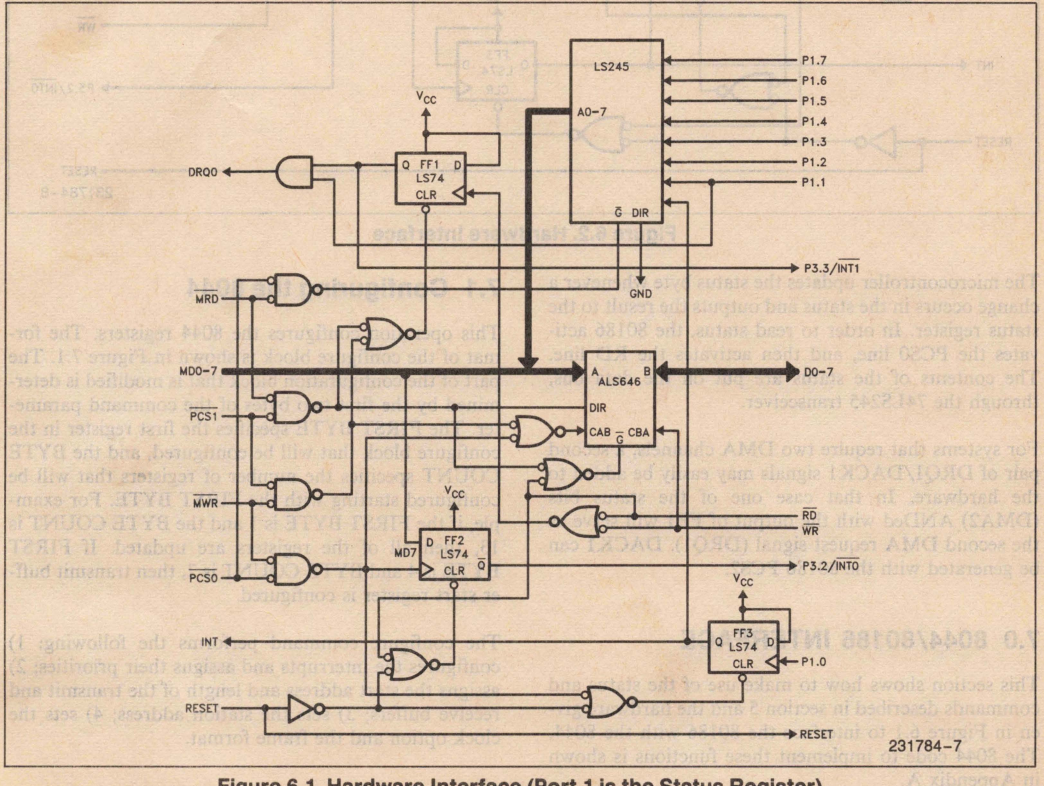


Figure 6.1. Hardware Interface (Port 1 is the Status Register)

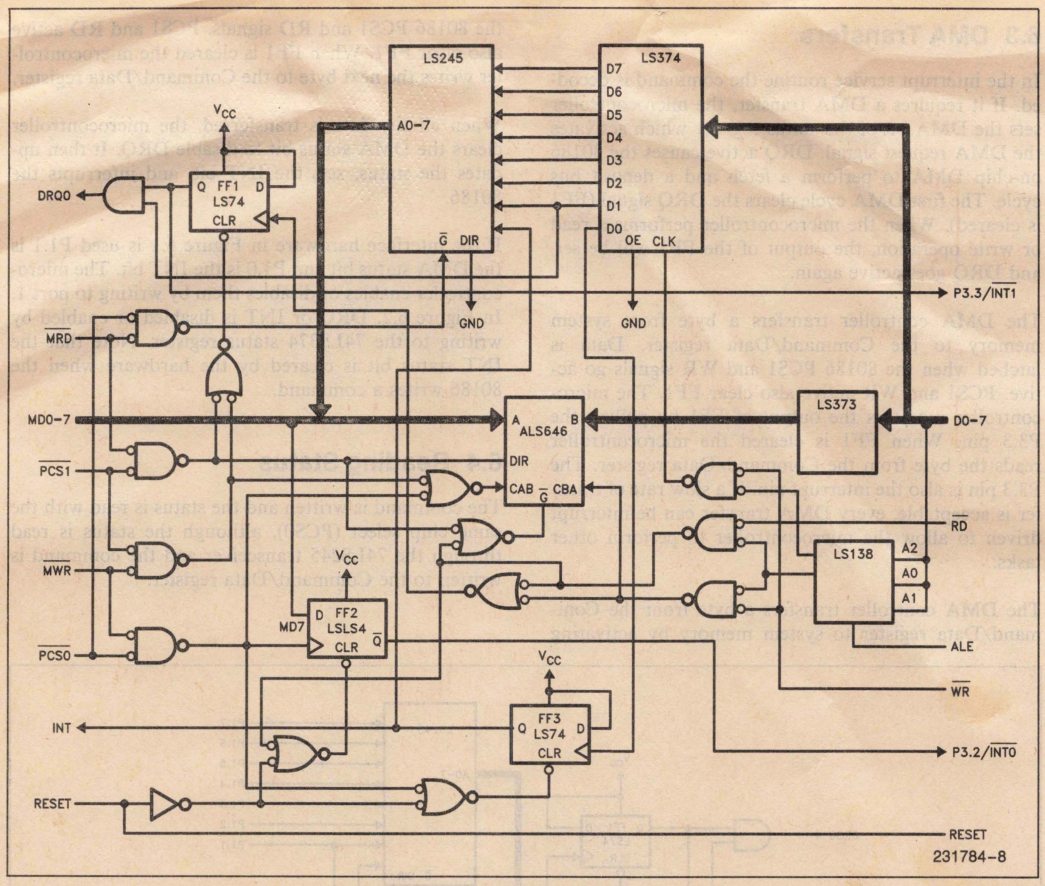


Figure 6.2. Hardware Interface

The microcontroller updates the status byte whenever a change occurs in the status and outputs the result to the status register. In order to read status, the 80186 activates the PCS0 line, and then activates the RD line. The contents of the status are put on the data bus, through the 74LS245 transceiver.

For systems that require two DMA channels, a second pair of DRQ1/DACK1 signals may easily be added to the hardware. In that case one of the status bits (DMA2) ANDed with the output of FF1 will serve as the second DMA request signal (DRQ1). DACK1 can be generated with the 80186 PCS2.

7.0 8044/80186 INTERFACE

This section shows how to make use of the status and commands described in section 5 and the hardware given in Figure 6.1 to interface the 80186 with the 8044. The 8044 code to implement these functions is shown in Appendix A.

7.1 Configuring the 8044

This operation configures the 8044 registers. The format of the configure block is shown in Figure 7.1. The part of the configuration block that is modified is determined by the first two bytes of the command parameter. The FIRST BYTE specifies the first register in the configure block that will be configured, and the BYTE COUNT specifies the number of registers that will be configured starting with the FIRST BYTE. For example, if the FIRST BYTE is 1 and the BYTE COUNT is 13, then all of the registers are updated. If FIRST BYTE is 4 and BYTE COUNT is 2, then transmit buffer start register is configured.

The configure command performs the following: 1) configures the interrupts and assigns their priorities; 2) assigns the start address and length of the transmit and receive buffers; 3) sets the station address; 4) sets the clock option and the frame format.

For other microcontrollers the format of the configure block should be modified accordingly. For example, the 80C51 serial port registers (e.g., T2CON, SCON) replace the 8044 SIU registers in the configure block.

7	6	5	4	3	2	1	0
BYTE COUNT							
FIRST BYTE							
STS							
SMD							
STATION ADDRESS							
TRANSMIT BUFFER START							
TRANSMIT BUFFER LENGTH							
RECEIVE BUFFER START							
RECEIVE BUFFER LENGTH							
INTERRUPT PRIORITY							
INTERRUPT ENABLE							
TIMER/COUNTER MODE							
TIMER/COUNTER MODE							
PROCESSOR STATUS WORD							

Figure 7.1. Format of the 8044 Configure Block

7.2 Transmitting a Message with the 8044

A message is a block of data which represents a text file or a set of instructions for a remote node or an application program which resides on the 8044 program memory. A message can be a frame (packet) by itself or can be comprised of multiple frames. An SDLC frame is the smallest block of data that the 8044 transmits. The 8044 can receive commands from the 80186 to transmit and receive messages. The 8044 on-chip CPU can be programmed to divide messages into frames if necessary. Maximum frame size is limited by the transmit or receive buffer.

To transmit a message, the 80186 prepares a transmit data block in memory as shown in Figure 7.2. Its first byte specifies the length of the rest of the block. The next two bytes specify the destination address of the node the message is being sent to and the control byte of the message. The 80186 programs the DMA controller with the start address of the block, length of the block and other control information and then issues the Transmit command to the 8044.

Upon receiving the command, the 8044 fetches the first byte of the block using DMA to determine the length of the rest of the block. It then fetches the destination address and the control byte using DMA.

The 8044 fetches the rest of the message into the on-chip transmit buffer. The size and location of the transmit buffer in the on-chip RAM is configured with the Configure command. The 8044 CPU then enables the Serial Interface Unit (SIU) to transmit the data as an SDLC frame. The SIU sends out the opening flag, the station address, the SDLC control byte, and the contents of transmit buffer. It then transmits the calculated CRC bytes and the closing flag. The 8044 CPU and the SIU operate concurrently. The CPU can fetch bytes from system memory or execute a command such as TRANSMIT-DISABLE while the SIU is active.

Upon completion of transmission, the SIU updates the internal registers and interrupts the 8044 CPU. The 8044 then updates the status and interrupts the 80186. Note that baud rate generation, zero bit insertion, NRZI encoding, and CRC calculation are automatically done by the SIU.

7.3 Receiving a Message with the 8044

To receive a message, the 80186 allocates a block of memory to store the message. It sets the DMA channel and sends the Receive command to the 8044.

Upon reception of the command, the 8044 enables its serial channel. The 8044 receives and passes to memory all frames whose address matches the individual or broadcast address and passes the CRC test.

The SIU performs NRZI decoding and zero bit deletion, then stores the information field of the received frame in the on-chip receive buffer. At the end of reception, the CPU requests the transfer of data bytes to 80186 memory using DMA. After transferring all the bytes, the 8044 transfers the data length, source address, and control byte of the received frame to the memory (see Figure 7.3). Upon completion of the transfers, the 8044 updates the status register and raises the interrupt signal to inform the 80186.

If the SIU is not ready when the first byte of the frame arrives, then the whole frame is ignored. Disabling reception after the first byte was passed to memory causes the rest of the frame to be ignored and an interrupt with Receive-Aborted event to be issued.

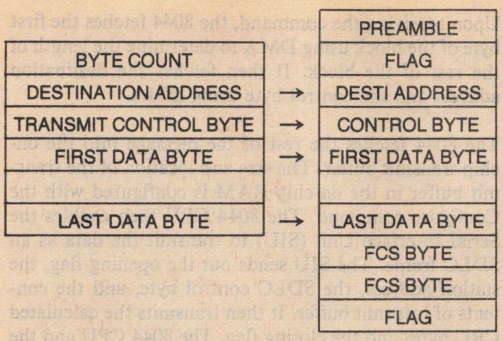


Figure 7.2. The 8044 Transmit Frame Structure and Location of Data Element in System Memory

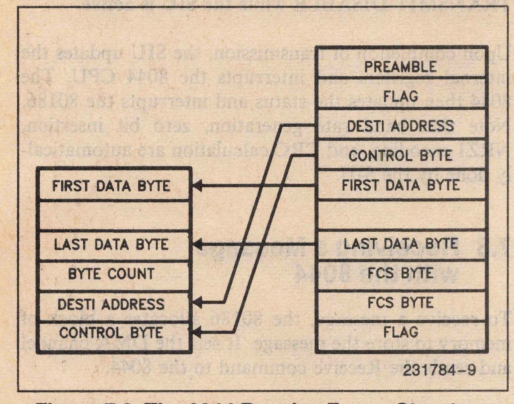


Figure 7.3. The 8044 Receive Frame Structure and Location of Received Data Element in System Memory

7.4 Dumping the 8044 Registers

Upon reception of the Dump command, the 8044 transfers the contents of its internal registers to the system memory (See Figure 7.4).

7 6 5 4 3 2 1 0
STS REG.
SMD REG.
STAD REG.
TBS REG.
TBL REG.
TCB REG.
RBS REG.
RBL REG.
RCB REG.
RFL REG.
PSW REG.
IP REG.
IE REG.
TMOD REG.
TCON REG.

Figure 7.4. Format of the 8044 Dumped Registers

7.5 Aborting an Operation

To abort a DMA operation, the 80186 sends an Abort command to the Command/Data latch and interrupts the 8044. During a DMA operation, the 8044 puts the external interrupt to high priority; therefore, the Abort interrupt will suspend the execution of the operation in progress and update the status register with the Execution-Aborted event. It then returns the 8044 program counter to a location before the aborted operation started. The Abort software procedure given in Appendix A gives the details of the execution of the ABORT command.

7.6 Disabling the Transmission or Reception

Transmission of a frame is aborted if the 80186 sends a TRANSMIT-DISABLE command to the 8044. The command causes the 8044 to clear the Transmit Buffer

Full (TBF) bit. During transmission, if the TBF bit is cleared, the SIU will discontinue the transmission and interrupt the 8044 CPU.

The RECEIVE-DISABLE command causes the 8044 to clear the Receive Buffer Empty (RBE) bit. The SIU aborts the reception, if the RBE bit is cleared by the CPU.

When transmission or reception of a frame is discontinued, the SIU interrupts the 8044 CPU. The CPU then updates the status and interrupts the 80186.

7.7 Handling Interrupts

When the 80186 sends a command, it sets the 8044 external interrupt flag. The 8044 services the interrupt at its own convenience. In the interrupt service routine the 8044 executes the appropriate instructions for a given command. During execution of a command the 8044 ignores any command, except ABORT, sent by the 80186 (see section 5.1.2). This is accomplished by clearing the interrupt flag before the 8044 returns from the interrupt service routine. During DMA operations the 8044 sets the external interrupt to high priority. An interrupt with high priority can suspend execution of an interrupt service routine with low priority. The ABORT command given by the 80186 will interrupt the execution of the DMA transfer in progress. Upon completion of ABORT, execution of the last operation will not be resumed (see Appendix A). Note that any other command given during the DMA operation will also abort the operation in progress and should be avoided.

8.0 8044 IN EXPANDED OPERATION

To increase the number of information bytes in a frame, the 8044 can be operated in Expanded mode. In Expanded operation the system memory can be used as the transmit and receive buffer instead of the 8044 internal RAM. AP-283, "Flexibility in Frame Size Operation with the 8044", describes Expanded operation in detail.

8.1 Transmitting a Message in Expanded Operation

In Expanded operation the 8044 transmits the frame while it is fetching the data from the system memory using DMA. An internal transmit buffer is not necessary. The system memory can be used as the transmit buffer by the 8044.

Upon receiving the Transmit command, the 8044 enables the SIU and fetches the first data byte from the Command/Data register. The SIU transmits the opening flag, station address, and the control byte if the frame format includes these fields. It then transmits the fetched data. The 8044 CPU fetches the next byte while the previously fetched byte is being transmitted by the SIU. The CPU fetches the remaining bytes using DMA, then the SIU transmits them simultaneously until the end of message is reached. The SIU then transmits the FCS bytes, the closing flag and interrupts the 8044 CPU. The 8044 updates the status with the Transmit-Done event and interrupts the 80186. If the DMA does not keep up with transmission, the transmission is an underrun.

8.2 Receiving a Message in Expanded Operation

In Expanded operation the DMA controller transfers data to the system memory while the 8044 SIU is receiving them.

To receive a message, the 80186 allocates a block of memory for storing the message. It sets the DMA channel and sends the Receive command to the 8044.

Upon reception of the command, the 8044 enables its serial channel and waits for a frame. The SIU performs flag detection, address filtering, zero bit deletion, NRZI decoding, and CRC checking as it does in Normal operation.

After the SIU receives the first byte of the frame, the 8044 CPU requests the transfer of the byte to memory using DMA. The 80186 DMA moves the information byte into the system memory while the SIU is receiving the next byte. The next byte is transferred to the memory after the SIU receives it. When the entire frame is received, the SIU checks the received Frame Check Sequence bytes. If there is no CRC error, the SIU updates the 8044 registers and interrupts the 8044 CPU. The CPU updates the status and interrupts the 80186.

9.0 CONCLUSION

This application note describes an efficient way to interface the 80186 and the 80188 microprocessors to the Intel 8-bit microcontrollers like the 80C51, 8052, and 8044. To illustrate this point the 80186 microprocessor interface to the 8044 microcontroller based serial communication chip was described. The hardware interface given here is very general and can interface the 8-bit microcontrollers to a variety of Intel microprocessors and DMA controllers. The microcontrollers with this interface hardware have the same benefits as both the Intel UPI-41/42 family and data communication peripheral chips such as the 82588 and the 82568 LAN controllers. Like the Intel UPI chips, they can be easily interfaced to microprocessors, and like the data communication peripherals, they execute high level commands. A similar approach can be used to interface Intel microprocessors to the 16-bit 8096 microcontroller.

APPENDIX A
SOFTWARE

The software modules shown here implement the execution of commands and status explained in sections 5 and 7. The 80186 software provides procedures to send commands and read status. The 8044 software decodes and executes the commands, updates the status, and interrupts the 80186. The procedures given here are called by higher level software drivers. For example, an 80186 application program may use the Transmit command to send a block of data to an application program that resides in the 8044 ROM or in another remote node. The application programs and the drivers that perform the communication tasks run asynchronously since all communication tasks are interrupt-driven.

Figure A-1 shows how to assign the ports and control registers for an 80186-based system. The software is written for an Intel iSBC® 186/51 computer board. The 8044 hardware is connected to the computer board iSBX™ connector.

Figure A-2 shows the 80186 command procedures. These procedures are used by the data link driver.

Figure A-3 shows how the DMA controller is loaded and initialized for data and parameter transfer from the 80186 memory to the 8044. This procedure is used by the TRANSMIT and CONFIGURE commands.

Figure A-4 shows how the DMA controller is loaded and initialized for data and parameter transfer from the 8044 to the 80186 memory. This procedure is used by the RECEIVE and DUMP commands.

Figure A-5 shows an interrupt service routine which handles interrupts resulting from various events. Note that this routine is not complete. The user should write the software to respond to events.

Figure A-6 shows an example of the 80186 software. It shows how to start various operations. This is not a data link driver, but it gives the procedures needed to write a complete driver.

Figure A-7 shows how to initialize the 8044. The user application program should be inserted here.

Figures A-8 through A-13 show the 8044 external interrupt service routine. In this routine a command received from the 80186 is decoded, and one of the command procedures shown in Figures A-9 through A-13 is executed.

Figure A-14 shows the serial channel (SIU) interrupt service routine. Note that execution of TRANSMIT, RECEIVE, and TRANSMIT-DISABLE commands are completed in this routine.

```
NAME COM_DRIVER

;**   80186 SOFTWARE FOR THE 80186/MICROCONTROLLER INTERFACE

;* 8044 BOARD CONNECTED TO THE SBX1 OF THE SBC 186/51 BOARD.
;* SBX1 INT0 TIED TO 80130 IR[0-7]. CONNECT JUMPER 30 TO 46.
;* 80186 DMA CHANNEL 1 USED. CONNECT JUMPER 202 TO 203.


TRUE          EQU   0FFFFH
FALSE         EQU   0H

; 8044 REGISTERS

CMD_44        EQU   080H            ; ADDRESS OF THE COMMAND REGISTER
ST_44         EQU   080H            ; ADDRESS OF THE STATUS REGISTER
DATA_44       EQU   0D4H            ; ADDRESS OF THE DATA REGISTER


; EVENTS

CON_DONE      EQU   01H             ; CONFIGURE_DONE
TRA_DONE      EQU   02H             ; TRANSMIT_DONE
DUM_DONE      EQU   03H             ; DUMP_DONE
REC_DONE      EQU   04H             ; RECEIVE_DONE
REC_DISA      EQU   05H             ; RECEPTION_DISABLE
TRA_DISA      EQU   06H             ; TRANSMISSION_DISABLE
ABO_DONE      EQU   07H             ; EXECUTION_ABORTED                    231784-10
; COMMANDS (INTA=1)

ABO_CMD       EQU   080H            ; ABORT
REC_DIS_CMD   EQU   081H            ; RECEIVE DISABLE
XMIT_DIS_CMD  EQU   082H            ; TRANSMIT DISABLE
REC_CMD       EQU   083H            ; RECEIVE
TRA_CMD       EQU   084H            ; TRANSMIT
DUM_CMD       EQU   085H            ; DUMP
CON_CMD       EQU   086H            ; CONFIGURE
NOP_CMD       EQU   087H            ; NOP

; 80186 DMA CHANNEL 1 REGISTERS

SL_DMA1       EQU   0FFD0H          ; SOURCE ADDRESS (LO WORD)
SH_DMA1       EQU   0FFD2H          ; SOURCE ADDRESS (HI WORD)
DL_DMA1       EQU   0FFD4H          ; DESTINATION ADDRESS (LO WORD)
DH_DMA1       EQU   0FFD6H          ; DESTINATION ADDRESS (HI WORD)
CNT_DMA1      EQU   0FFD8H          ; TRANSFER COUNT ADDRESS
CTL_DMA1      EQU   0FFDAH          ; CONTROL ADDRESS

; 80186 INTERRUPT CONTROLLER REGISTERS

CTL0_INTR     EQU   0FF38H          ; INT 0 CONTROL ADDRESS
CTL1_INTR     EQU   0FF3AH          ; INT 1 CONTROL REGISTER
MASK_INTR     EQU   0FF28H          ; INT MASK REGISTER
EOI_INTR      EQU   0FF22H          ; INT EOI REGISTER
NSPEC_BIT     EQU   08000H          ; NON-SPECIFIC EOI

; 80130 INTERRUPT CONTROLLER REGISTERS

EOI_SINTR     EQU   0E0H            ; INT EOI REGISTER
MASK_SINTR    EQU   0E2H            ; MASK REGISTER

RD_IRR        EQU   010H            ; COMMAND TO 80130 TO READ IRR REG
RD_ISR        EQU   011H            ; COMMAND TO 80130 TO READ ISR REG


IV_BASE       EQU   20H             ; BASE OF 80130 INT CONTROLLER VECTOR
                                                                     231784-11
```

Figure A-1. Port and Register Definitions for 80186 System

```
;*****************************************************************
; INTERRUPT TABLE

INTERRUPTS      SEGMENT  AT 0

                ORG    (IV_BASE+1)*4H

IV_INTR0        LABEL    DWORD          ; IR1 VECTOR

INTERRUPTS      ENDS

;*****************************************************************

STACK           SEGMENT    STACK  'STACK'

THE_STACK       DW       200H     DUP(?)
TOS     LABEL            WORD

STACK           ENDS

;*****************************************************************

DATA            SEGMENT    PUBLIC  'DATA'

REC_BUFFER       DB        1024    DUP(?)

CON_BUFFER       DB        08H,01H,00H,0D0H,55H,20H,05H,30H,05H

DUM_BUFFER       DB        0FH     DUP(?)

TRA_BUFFER       DB        07H,55H,11H,01H,02H,03H,04H,05H

CMND_FLAG        DW        FALSE

DATA            ENDS
```

231784-12

Figure A-1. Port and Register Definitions for 80186 System (Continued)

```
;*************************************************************

CODE        SEGMENT  PUBLIC  'CODE'

ASSUME      CS:CODE,
&           DS:DATA,
&           ES:NOTHING,
&           SS:STACK

;*************************************************************

RECV_COMMAND     PROC   FAR

      PUSH    BP
      MOV     BP,SP
      LES     SI,DWORD PTR [BP+6]      ; LOAD BUFFER POINTER
      MOV     AX,WORD PTR[BP+10]       ; LOAD BUFFER SIZE
      MOV     AH,0H
      CALL    REC_DMA                  ; CALL REC-DMA
      MOV     AL,REC_CMD               ; LOAD RECEIVE COMMAND
      OUT     CMD_44,AL                ; SEND TO COMMAND/DATA REG
      POP     BP
      RET

RECV_COMMAND     ENDP

;*************************************************************

XMIT_COMMAND    PROC     FAR

      PUSH    BP
      MOV     BP,SP
      LES     SI,DWORD PTR [BP+6]      ; LOAD BUFFER POINTER
      MOV     AX,WORD PTR[BP+10]       ; LOAD BUFFER SIZE
      MOV     AH,0H
      CALL    TRA_DMA                  ; CALL TRA-DMA
      MOV     AL,TRA_CMD               ; LOAD TRANSMIT COMMAND
      OUT     CMD_44,AL                ; SEND TO COMMAND/DATA REG
      POP     BP
      RET

XMIT_COMMAND    ENDP
```

231784-13

Figure A-2. Setup and Execution of Commands

```
;*****************************************************************

CONF_COMMAND    PROC    FAR

      PUSH    BP
      MOV     BP,SP
      LES     SI,DWORD PTR[BP+6]      ; LOAD BUFFER POINTER
      MOV     AX,WORD PTR[BP+10]      ; LOAD BUFFER SIZE
      MOV     AH,0H
      CALL    TRA_DMA                 ; CALL TRA-DMA
      MOV     AL,CON_CMD              ; LOAD CONFIGURE COMMAND
      OUT     CMD_44,AL               ; SEND TO COMMAND/DATA REG
      POP     BP
      RET

CONF_COMMAND    ENDP

;*****************************************************************

DUMP_COMMAND    PROC    FAR

      PUSH    BP
      MOV     BP,SP
      LES     SI,DWORD PTR[BP+6]      ; LOAD BUFFER POINTER
      MOV     AX,WORD PTR[BP+10]      ; LOAD BUFFER SIZE
      MOV     AH,0H
      CALL    REC_DMA                 ; CALL REC-DMA
      MOV     AL,DUM_CMD              ; LOAD DUMP COMMAND
      OUT     CMD_44,AL               ; SEND TO COMMAND/DATA REG
      POP     BP
      RET

DUMP_COMMAND    ENDP                                        231784-14
;*****************************************************************

XMIT_DIS_COMMAND    PROC    FAR

      MOV     AL,XMIT_DIS_CMD         ; LOAD XMIT-DIS  COMMAND
      OUT     CMD_44,AL               ; SEND TO COMMAND/DATA REG
      RET

XMIT_DIS_COMMAND    ENDP

;*****************************************************************

REC_DIS_COMMAND     PROC    FAR

      MOV     AL,REC_DIS_CMD          ; LOAD REC-DIS  COMMAND
      OUT     CMD_44,AL               ; SEND TO COMMAND/DATA REG
      RET

REC_DIS_COMMAND     ENDP

;*****************************************************************

ABOR_COMMAND    PROC    FAR

      MOV     AL,ABO_CMD              ; LOAD ABORT COMMAND
      OUT     CMD_44,AL               ; SEND TO COMMAND/DATA REG
      RET

ABOR_COMMAND    ENDP

;*****************************************************************

NOP_COMMAND     PROC    FAR

      MOV     AL,NOP_CMD              ; LOAD NOP COMMAND
      OUT     CMD_44,AL               ; SEND TO COMMAND/DATA REG
      RET

NOP_COMMAND     ENDP                                        231784-15
```

Figure A-2. Setup and Execution of Commands (Continued)

```
;*************************************************************
; **  RECEIVE DMA
; ARGS      AX      BUFFER SIZE
;           ES:SI   BUFFER POINTER

REC_DMA     PROC        NEAR
     MOV    DX,CNT_DMA1             ; LOAD ADD OF TRANSFER COUNT REG
     OUT    DX,AX                   ; PROGRAM TRANSFER COUNT REGISTER

     XOR    BX,BX                   ; CLEAR BX
     MOV    AX,ES                   ; LOAD SEG ADDRESS OF BUFFER
     SHL    AX,1                    ; CALCULATE LINEAR ADDRESS OF THE BUFFER
     RCL    BX,1
     SHL    AX,1
     RCL    BX,1
     SHL    AX,1
     RCL    BX,1
     SHL    AX,1
     RCL    BX,1
     ADD    AX,SI                   ; ADD THE OFFSET TO BASE
     ADC    BX,0
     MOV    DX,DL_DMA1              ; LOAD ADDRESS OF DEST POINTER (LO WORD)
     OUT    DX,AX                   ; PROGRAM DEST POINTER REGISTER (LO WORD)
     MOV    AX,BX
     MOV    DX,DH_DMA1              ; LOAD ADDRESS OF DEST POINTER (HI WORD)
     OUT    DX,AX                   ; PROGRAM DEST POINTER REGISTER (HI WORD)

     MOV    AX,DATA_44              ; LOAD ADDRESS OF DATA REGISTER
     MOV    DX,SL_DMA1              ; LOAD ADDRESS OF SOURCE POINTER
     OUT    DX,AX                   ; PROGRAM SOURCE POINTER REGISTER (LO WORD)

     XOR    AX,AX                   ; CLEAR AX
     MOV    DX,SH_DMA1              ; LOAD ADDRESS OF SOURCE POINTER (HI WORD)
     OUT    DX,AX                   ; PROGRAM SOURCE POINTER REGISTER (HI WORD)

     MOV    DX,CTL_DMA1             ; LOAD ADDRESS OF CONTROL REGISTER
     MOV    AX,1010001010100110B    ; LOAD THE CONTROL WORD
     OUT    DX,AX                   ; PROGRM THE CONTRL REGISTER
     RET

REC_DMA     ENDP
```

231784-16

Figure A-3. Loading and Starting the 80186 DMA Controller

```
;*****************************************************************
; ** TRANSMIT DMA
; ARGS  AX        BUFFER SIZE
;       ES:SI     BUFFER POINTER
;
TRA_DMA     PROC    NEAR
     INC    AX
     MOV    DX,CNT_DMA1              ; LOAD ADD OF TRANSFER COUNT REG
     OUT    DX,AX                  ; PROGRAM TRANSFER COUNT REGISTER

     XOR    BX,BX                   ; CLEAR BX
     MOV    AX,ES                   ; LOAD SEG ADDRESS OF BUFFER
     SHL    AX,1                    ; CALCULATE LINEAR ADDRESS OF THE BUFFER
     RCL    BX,1
     SHL    AX,1
     RCL    BX,1
     SHL    AX,1
     RCL    BX,1
     SHL    AX,1
     RCL    BX,1
     ADD    AX,SI                   ; ADD THE OFFSET TO BASE
     ADC    BX,0
     MOV    DX,SL_DMA1              ; LOAD ADDRESS OF SOURCE POINTER (LO WORD)
     OUT    DX,AX                   ; PROGRAM SOURCE POINTER REGISTER (LO WORD)
     MOV    AX,BX
     MOV    DX,SH_DMA1              ; LOAD ADDRESS OF SOURCE POINTER (HI WORD)
     OUT    DX,AX                   ; PROGRAM SOURCE POINTER REGISTER (HI WORD)

     MOV    AX,DATA_44              ; LOAD ADDRESS OF DATA REGISTER
     MOV    DX,DL_DMA1              ; LOAD ADDRESS OF DEST POINTER
     OUT    DX,AX                   ; PROGRAM DEST POINTER REGISTER (LO WORD)

     XOR    AX,AX                   ; CLEAR AX
     MOV    DX,DH_DMA1              ; LOAD ADDRESS OF DEST POINTER (HI WORD)
     OUT    DX,AX                   ; PROGRAM DEST POINTER REGISTER (HI WORD)

     MOV    DX,CTL_DMA1             ; LOAD ADDRESS OF CONTROL REGISTER
     MOV    AX,0001011010100110B    ; LOAD THE CONTROL WORD
     OUT    DX,AX                   ; PROGRAM THE CONTRL REGISTER
     RET

TRA_DMA     ENDP
```

231784-17

Figure A-4. Loading and Starting the 80186 DMA Controller

```
;*************************************************************************
;  80186 INTERRUPT ROUTINE

INT_186:

     PUSH  AX
     PUSH  DX
     MOV   AX,NSPEC_BIT                ; SEND NSPEC END OF INT
     MOV   DX,EOI_INTR
     OUT   DX,AX

     MOV   AL,01100001B
     OUT   EOI_SINTR,AL

     IN    AL,ST_44                    ; READ THE STATUS
     AND   AX,0FFH

; DECODE STATUS AND TAKE APPROPRIATE ACTION

     MOV   DX,CTL_DMA1                 ; DISABLE DMA
     IN    AX,DX
     OR    AX,0100B
     AND   AX,NOT 010B
     OUT   DX,AX

     MOV   CMND_FLAG,TRUE

     POP   DX
     POP   AX
     IRET
```

231784-18

Figure A-5. Interrupt Service Routine

```
;*************************************************************

BEGIN:
        CLI
        CLD

; SET ALL REGISTERS SMALL MODEL

          MOV     SP,DATA
          MOV     DS,SP
          MOV     ES,SP
          MOV     SP,STACK
          MOV     SS,SP
          MOV     SP,OFFSET TOS

; SETUP INTERRUPT VECTORS

          PUSH    ES
          XOR     AX,AX
          MOV     ES,AX
          MOV     WORD PTR ES:IV_INTR0 +0, OFFSET INT_186
          MOV     WORD PTR ES:IV_INTR0 +2, CS
          POP     ES

SETUP 80130 INTERRUPT CONTROLLER

          MOV     AL,00010011B          ; ICW1
          OUT     EOI_SINTR,AL
          MUL     AL

          MOV     AL,IV_BASE            ; ICW2
          OUT     MASK_SINTR,AL
          MUL     AL

          MOV     AL,00000000B          ; ICW4
          OUT     MASK_SINTR,AL
          MUL     AL

          MOV     AL,0FCH               ;MASK
          OUT     MASK_SINTR,AL
```

231784-19

Figure A-6. Example of Executing Commands

```
; SETUP 80186 INTERRUPT CONTROLLER

        MOV     AX,0000000000100000B
        MOV     DX,CTL0_INTR
        OUT     DX,AX

        MOV     DX,CTL1_INTR
        IN      AX,DX
        OR      AX,0000000000101000B
        OUT     DX,AX

        MOV     AX,000EDH               ; MASK ALL BUT I0
        MOV     DX,MASK_INTR
        OUT     DX,AX
        STI                             ;ENABLE INTERRUPTS

;***  SEND CONFIURE COMMAND

        PUSH    WORD PTR CON_BUFFER     ; PUSH BUFFER SIZE
        PUSH    DS                      ; PUSH BUFFER SEGMENT REGISTER
        PUSH    OFFSET CON_BUFFER       ; PUSH  OFFSET OF BUFFER
        CALL    CONF_COMMAND            ; CALL CONFIGURE
        ADD     SP,3*2

; WAIT FOR END OF COMMAND

WAIT1:
        CMP     CMND_FLAG,TRUE
        JNE     WAIT1
        MOV     CMND_FLAG,FALSE
```

231784-20

```
;***  SEND DUMP COMMAND

        PUSH    WORD PTR DUM_BUFFER     ; PUSH BUFFER SIZE
        PUSH    DS                      ; PUSH BUFFER SEGMENT REGISTER
        PUSH    OFFSET DUM_BUFFER       ; PUSH  OFFSET OF BUFFER
        CALL    DUMP_COMMAND            ; CALL CONFIGURE
        ADD     SP,3*2

WAIT2:
        CMP     CMND_FLAG,TRUE
        JNE     WAIT2
        MOV     CMND_FLAG,FALSE

;*** SEND TRANSMIT COMMAND

        PUSH    WORD PTR TRA_BUFFER     ; PUSH BUFFER SIZE
        PUSH    DS                      ; PUSH BUFFER SEGMENT REGISTER
        PUSH    OFFSET TRA_BUFFER       ; PUSH  OFFSET OF BUFFER
        CALL    XMIT_COMMAND            ; CALL COMMAND
        ADD     SP,3*2

WAIT3:
        CMP     CMND_FLAG,TRUE
        JNE     WAIT3
        MOV     CMND_FLAG,FALSE

;*** SEND RECEIVE COMMAND

        PUSH    WORD PTR REC_BUFFER     ; PUSH BUFFER SIZE
        PUSH    DS                      ; PUSH BUFFER SEGMENT REGISTER
        PUSH    OFFSET REC_BUFFER       ; PUSH  OFFSET OF BUFFER
        CALL    RECV_COMMAND            ; CALL COMMAND
        ADD     SP,3*2

WAIT4:
        CMP     CMND_FLAG,TRUE
        JNE     WAIT4
        MOV     CMND_FLAG,FALSE


CODE            ENDS
        END     BEGIN
```

231784-21

Figure A-6. Example of Executing Commands (Continued)

```
$DEBUG NOMOD51
$INCLUDE (REG44.PDF)

; THE 8044 SOFTWARE DRIVER FOR THE 80186/8044 INTERFACE.

        ORG   00H               ; LOCATIONS 00 THRU 26H ARE USED
        SJMP  INIT              ; BY INTERRUPT SERVICE ROUTINES.
        ORG   03H               ; VECTOR ADDRESS FOR EXT INT0.
        JMP   EINT0
        ORG   23H               ; VECTOR ADDRESS FOR SERIAL INT
        JMP   SIINT

;********************* INITIALIZATION *************************

        ORG   26H
INIT:   MOV   TCON,#00000001B  ; EXT INT0: EDGE TRIGGER
        MOV   IE,#00010001B    ; SI=EX0=1
        CLR   P1.1             ; CLEAR DRQ STATUS BIT
        SETB  EA               ; ENABLE INTERRUPTS
DOT:    SJMP  DOT              ; WAIT FOR AN INTERRUPT
                                                        231784-22
```

Figure A-7. Initialization Routine

```
;******************EXTERNAL INTERRUPT 0 *********************
EINT0:  CLR   P1.5              ; CLEAR THE E BIT
        MOV   DPTR,#100H        ; LOAD DATA POINTER WITH A DUMY NUMBER
        MOVX  A,@DPTR           ; READ THE COMMAND BYTE.
        ANL   A,#00001111B      ; KEEP THE OPERATION FIELD
        MOV   R2,A              ; SAVE COMMAND

; DECODE COMMAND AND JUMP TO THE APPROPRIATE ROUTINE
;        COMMAND          OPERATION (BITS0-3)
;
;          ABORT              00H
;          REC-DISABLE        01H
;          TRA-DISABLE        02H
;          RECEIVE            03H
;          TRANSMIT           04H
;          DUMP               05H
;          CONFIGURE          06H
;          NOP                07H

        JNB    PX0,J1           ; IF INT0 IS SET TO PRIORITY 1,
        JMP    CABO             ;THEN DMA OPERATION WAS IN PROGRESS.
                                ; EXECUTE ABORT REGARDLESS OF THE
                                ;COMMAND ISSUED.
J1:     CJNE  A,#00H,J2         ; EXECUTE ABORT
        JMP   CABO              ; THIS LINE WILL BE EXECUTED IF ABORT WAS
                                ;ISSUED WHEN THE 8044 IS NOT EXECUTING
                                ;ANY COMMANDS.
J2:     CJNE  A,#01H,J3
        JMP   CRDIS             ; EXECUTE RECEIVE-DISCONNECT
J3:     CJNE  A,#0B5H,J4
        JMP   CTDIS             ; EXECUTE TRANSMIT-DISCONNECT
J4:     CJNE  A,#03H,J5
        JMP   CREC              ; EXECUTE RECEIVE
J5:     CJNE  A,#04H,J6
        JMP   CTRA              ; EXECUTE TRANSMIT
J6:     CJNE  A,#05H,J7
        JMP   CDUMP             ; EXECUTE DUMP
J7:     CJNE  A,#06H,J8
        JMP   CCON              ; EXECUTE CONFIGURE
J8:     CJNE  A,#07H,J9
        JMP   CNOP              ; EXECUTE NOP
J9:     RETI                    ; RETURN. OPERATION NOT RECOGNIZED.
                                                        231784-23
```

Figure A-8. External Interrupt Service Routine

```
; **  NOP COMMAND

CNOP:    CLR    IE0              ; IGNORE PENDING EXT INT0 (IF ANY).
                                 ; ANY INTERRUPT (COMMNAD) DURING
                                 ; EXECUTION OF AN OPERATION IS IGNORED
         RETI                    ; RETURN

; **  ABORT COMMAND

CABO:    JNB    PX0,CABOJ1       ; WAS DMA IN PROGRESS?
         CLR    PX0              ; YES. EXT INT0: PRIORITY 0
         CLR    P1.1             ; CLEAR DMA REQUEST

         SETB   P1.2             ; UPDATE STATUS WITH
         SETB   P1.3             ;ABORT-DONE EVENT
         SETB   P1.4             ; (STATUS=DDH; E=0)

         CLR    IE0              ; IGNORE PENDING EXT INT0 (IF ANY).
         CLR    P1.0
         SETB   P1.0             ; SET INT BIT AND INTERRUPT 80186
         JB     P3.2,$           ; WAIT TILL INTERRUPT IS ACKNOWLEDGED
                                 ; EXECUTE THE NEXT "RETI" TWICE
         POP    ACC              ; POP OUT THE OLD HI BYTE PC
         POP    ACC              ; POP OUT THE OLD LOW BYTE PC
         MOV    B,#HIGH($+10)    ; HI BYTE ADDRESS OF CABOJ2
         MOV    ACC,#LOW($+7)    ; LOW BYTE ADDRESS OF CABOJ2
         PUSH   ACC              ; PUSH THE ADDRESS OF THE NEXT
         PUSH   B                ;"RETI" INSTRUCTION INTO STACK
CABOJ2:  RETI                    ; RETURN

CABOJ1:  NOP                     ; DMA WAS NOT IN PROGRESS
         SETB   P1.5             ; SET THE E BIT

         SETB   P1.2             ; UPDATE STATUS WITH
         SETB   P1.3             ;ABORT-DONE EVENT
         SETB   P1.4             ; (STATUS=FDH; E=1)

         CLR    IE0              ; IGNORE PENDING EXT INT0 (IF ANY).
         CLR    P1.0
         SETB   P1.0             ; SET INT BIT AND INTERRUPT 80186
         JB     P3.2,$           ; WAIT TILL INTERRUPT IS ACKNOWLEDGED
         RETI                    ; RETURN
```

231784-24

Figure A-9. Execution of NOP and ABORT Commands

```
; ** CONFIGURE COMMNAD

CCON:    MOV    DPTR,#100H
         CLR    IE0              ; IGNORE PENDING EXT INT0 (IF ANY)
         SETB   PX0              ; EXT INT0: PRIORITY 1
                                 ; PX0 IS SET TO ACCEPT ABORT
                                 ;DURING DMA OPERATION.
         SETB   P1.1             ; ENABLE DMA REQUEST
         JB     P3.3,$           ; WAIT FOR DMA ACK.
         MOVX   A,@DPTR          ; READ FROM COMMAN/DATA REGISTER
         MOV    R0,A             ; LOAD BYTE COUNT
         DEC    R0               ; DECREMENT BYTE COUNT
         JB     P3.3,$           ; WAIT FOR DMA ACK.
         MOVX   A,@DPTR          ; READ FROM COMMAND/DATA REGISTER
         MOV    R1,A             ; LOAD FIRST-BYTE
         JB     P3.3,$           ; WAIT FOR DMA ACK.
         MOVX   A,@DPTR          ; READ FROM COMMAND/DATA REGISTER
         CJNE   R1,#01H,CCONJ1   ; CHECK THE FIRST-BYTE
         MOV    STS,A            ; UPDATE THE STS REGISTER
         INC    R1               ; INC. POINTER TO THE CONF. BLOCK
         DJNZ   R0,CCONF4        ; CHECK THE BYTE COUNT
         JMP    CCONT1
CCONF4:  JB     P3.3,CCONF4
         MOVX   A,@DPTR
CCONJ1:  CJNE   R1,#02H,CCONJ2
         MOV    SMD,A
         INC    R1
         DJNZ   R0,CCONF5
         JMP    CCONT1
CCONF5:  JB     P3.3,CCONF5
         MOVX   A,@DPTR
CCONJ2:  CJNE   R1,#03H,CCONJ3
         MOV    STAD,A
         INC    R1
         DJNZ   R0,CCONF6
         JMP    CCONT1
CCONF6:  JB     P3.3,CCONF6
         MOVX   A,@DPTR
CCONJ3:  CJNE   R1,#04H,CCONJ4
```

231784-25

Figure A-10. Execution of CONFIGURE Command

```
                MOV   TBS,A
                INC   R1
                DJNZ  R0,CCONF7
                JMP   CCONT1
CCONF7:         JB    P3.3,CCONF7
                MOVX  A,@DPTR
CCONJ4:         CJNE  R1,#05H,CCONJ5
                MOV   TBL,A
                INC   R1
                DJNZ  R0,CCONF8
                JMP   CCONT1
CCONF8:         JB    P3.3,CCONF8
                MOVX  A,@DPTR
CCONJ5:         CJNE  R1,#06H,CCONJ6
                MOV   RBS,A
                INC   R1
                DJNZ  R0,CCONF9
                JMP   CCONT1
CCONF9:         JB    P3.3,CCONF9
                MOVX  A,@DPTR
CCONJ6:         CJNE  R1,#07H,CCONJ7
                MOV   RBL,A
                INC   R1
                DJNZ  R0,CCONFA
                JMP   CCONT1
CCONFA:         JB    P3.3,CCONFA
                MOVX  A,@DPTR
CCONJ7:         CJNE  R1,#08H,CCONJ8
                MOV   IP,A
                INC   R1
                DJNZ  R0,CCONFB
                JMP   CCONT1
CCONFB:         JB    P3.3,CCONFB
                MOVX  A,@DPTR
CCONJ8:         CJNE  R1,#09H,CCONJ9
                MOV   IE,A
                INC   R1
                DJNZ  R0,CCONFC
                JMP   CCONT1                                    231784-26
CCONFC:         JB    P3.3,CCONFC
                MOVX  A,@DPTR
CCONJ9:         CJNE  R1,#0AH,CCONJA
                MOV   TMOD,A
                INC   R1
                DJNZ  R0,CCONFD
                JMP   CCONT1
CCONFD:         JB    P3.3,CCONFD
                MOVX  A,@DPTR
CCONJA:         CJNE  R1,#0BH,CCONJB
                MOV   TCON,A
                INC   R1
                DJNZ  R0,CCONFE
                JMP   CCONT1
CCONFE:         JB    P3.3,CCONFE
                MOVX  A,@DPTR
CCONJB:         CJNE  R1,#0CH,ERROR1
                MOV   PSW,A
                INC   R1
                DJNZ  R0,ERROR1
                JMP   CCONT1

ERROR1:         NOP                     ; ILLEGAL BYTE COUNT
                SETB  P1.5              ; SET THE E STATUS BIT

CCONT1:         NOP
                CLR   P1.1              ; CLEAR DMA REQUEST
                CLR   PX0               ; EXT INT0: PRIORITY 0

                SETB  P1.2              ; UPDATE STATUS WITH
                CLR   P1.3              ;CONFIGURE-DONE EVENT
                CLR   P1.4              ; (STATUS=C5H IF E=0)

                CLR   IE0               ; IGNORE PENDING EXT INT0 (IF ANY)
                CLR   P1.0
                SETB  P1.0              ; INTERRUPT THE 80186
                JB    P3.2,$            ; WAIT TILL INTERRUPT IS ACKNOWLEDGED
                RETI                    ; RETURN
                                                                231784-27
```

Figure A-10. Execution of CONFIGURE Command (Continued)

```
; ** DUMP COMMAND

CDUMP:  MOV   A,STS             ; LOAD THE FIRST DUMP REG INTO ACC
        MOVX  @DPTR,A           ; WRITE TO THE COMMAND/DATA REGISTER
        CLR   IE0               ; IGNORE PENDING EXT INT0 (IF ANY)
        SETB  PX0               ; INTRERRUPT 0: PRIORITY 1
        SETB  P1.1              ; ENABLE DMA REQUEST
        JB    P3.3,$            ; WAIT FOR DMA ACK
        MOV   A,SMD
        MOVX  @DPTR,A
        JB    P3.3,$
        MOV   A,STAD
        MOVX  @DPTR,A
        JB    P3.3,$
        MOV   A,TBS
        MOVX  @DPTR,A
        JB    P3.3,$
        MOV   A,TBL
        MOVX  @DPTR,A
        JB    P3.3,$
        MOV   A,TCB
        MOVX  @DPTR,A
        JB    P3.3,$
        MOV   A,RBS
        MOVX  @DPTR,A
        JB    P3.3,$
        MOV   A,RBL
        MOVX  @DPTR,A
        JB    P3.3,$
        MOV   A,RCB
        MOVX  @DPTR,A
        JB    P3.3,$
        MOV   A,RFL
        MOVX  @DPTR,A
        JB    P3.3,$
        MOV   A,PSW
        MOVX  @DPTR,A
        JB    P3.3,$
        MOV   A,IP                                        231784-28
        MOVX  @DPTR,A
        JB    P3.3,$
        MOV   A,IE
        MOVX  @DPTR,A
        JB    P3.3,$
        MOV   A,TMOD
        MOVX  @DPTR,A
        JB    P3.3,$
        MOV   A,TCON
        MOVX  @DPTR,A
        JB    P3.3,$
        CLR   P1.1              ; DISABLE DRQ
        CLR   PX0               ; EXTERNAL INT0: PRIORITY 0

        SETB  P1.2              ; UPDATE STATUS WITH
        SETB  P1.3              ;DUMP-DONE EVENT
        CLR   P1.4              ; (STATUS=CDH)

        CLR   IE0               ; IGNORE PENDING EXT INT0
        CLR   P1.0
        SETB  P1.0              ; INTERRUPT THE 80186
        JB    P3.2,$            ; WAIT TILL INTERRUPT IS ACKNOWLEDGED
        RETI                    ; RETURN
                                                          231784-29
```

Figure A-11. Execution of DUMP Command

```
; ** RECEIVE COMMAND.
CREC:    JNB   RBE,CRECJ1        ; IS SIU ALREADY IN RECEIVE MODE?
         SETB  P1.5              ; YES. SET THE E BIT
CRECJ1:  SETB  RBE               ; NO. ENABLE RECEPTION
         CLR   RBP               ; CLEAR RECEIVE BUFFER PROTECT BIT
         CLR   IE0               ; IGNORE PENDING EXT INT0 (IF ANY)
         RETI                    ; RETURN. UPDATE STATUS IN THE
                                 ;SIU INTERRUPT ROUTINE.

; ** TRANSMIT COMMAND.
CTRA:    MOV   R1,TBS             ; LOAD TRANSMIT BUFFER START
         CLR   IE0                ; IGNORE PENDING EXT INT0 (IF ANY)
         SETB  PX0                ; EXT INT0: PRIROITY 1
         SETB  P1.1               ; ENABLE DMA REQUEST
         JB    P3.3,$             ; WAIT FOR DMA ACK.
         MOVX  A,@DPTR            ; READ FROM COMMAND/DATA REG.
         MOV   R0,A               ; LOAD THE BYTE COUNT
         DEC   A                  ; SUBTRACT 2 FROM THE BYTE
         DEC   A                  ;COUNT AND LOAD INTO XMIT
         MOV   TBL,A              ; LOAD BUFFER LENGTH
CTRAJ2:  JB    P3.3,CTRAJ2        ; WAIT FOR DMA ACK.
         MOVX  A,@DPTR            ; READ FROM COMMAND/DATA REG.
         MOV   STAD,A             ; LOAD DESTINATION ADDRESS
         DEC   R0                 ; DECREMENT THE BYTE COUNT
CTRAJ3:  JB    P3.3,CTRAJ3        ; WAIT FOR DMA ACK.
         MOVX  A,@DPTR            ; READ FROM COMMAND/DATA REG.
         MOV   TCB,A              ; LOAD THE TRANSMIT CONTROL BYTE
         DJNZ  R0,CTRAJ4          ; IS THERE ANY INFO. BYTE?
         SJMP  CTRAJ5             ; NO.
CTRAJ4:  JB    P3.3,CTRAJ4        ; YES. WAIT FOR DMA ACK.
         MOVX  A,@DPTR            ; READ FROM COMMAND/DATA REG.
         MOV   @R1,A              ; MOVE DATA TO THE TRANSMIT BUFFER
         INC   R1                 ; INC. POINTER TO BUFFER
         DJNZ  R0,CTRAJ4          ; LAST BYTE FETCHED INTO THE BUFFER?
                                  ; NO. FETCH THE NEXT BYTE
CTRAJ5:  CLR   P1.1               ; YES. DISABLE DMA REQUEST
         CLR   PX0                ; EXT INT0: PRIORITY 0
         SETB  TBF                ; SET TRANSMIT BUFFER FULL
         SETB  RTS                ; ENABLE TRANSMISSION
         CLR   IE0                ; IGNORE PENDING EXT INT0 (IF ANY)
         RETI                     ; RETURN. UPDATE STATUS IN THE
                                  ;SIU INTERRUPT ROUTINE
```

231784-30

Figure A-12. Execution of RECEIVE and TRANSMIT Commands

```
; ** TRANSMIT-DISCONNECT COMMAND

CTDIS:   JB    TBF,CTDIJ1         ; IS TRANSMIT BUFFER ALREADY EMPTY?
         SETB  P1.5               ; YES, SET THE E BIT
CTDIJ1:  CLR   TBF                ; NO. CLEAR TRANSMIT BUFFER
         CLR   IE0                ; IGNORE PENDING EXT INT0 (IF ANY)
         RETI                     ; RETURN. UPATE STATUS IN THE
                                  ;SIU INTERRUPT ROUTINE.



; ** RECEIVE-DISCONNECT COMMAND

CRDIS:   JB    RBE,CRDIJ1         ; IS RECEIVE BUFFER ALREADY EMPTY?
         SETB  P1.5               ; YES. SET THE E BIT
CRDIJ1:  CLR   RBE                ; NO. CLEAR RECEIVE BUFFER

         SETB  P1.2               ; UPDATE STATUS WITH
         CLR   P1.3               ;RECEPTION-DISABLED EVENT
         SETB  P1.4               ; (STATUS=D5 IF E=0)

         CLR   IE0
         CLR   P1.0
         SETB  P1.0               ; INTERRUPT THE 80186
         JB    P3.2,$             ; WAIT TILL INTERRUPT IS ACKNOWLEDGED
         RETI                     ; RETURN
```

231784-31

Figure A-13. Execution of RECEIVE-DISCONNECT and TRANSMIT-DISCONNECT Commands

```
;************ SERIAL CHANNEL (SIU) INTERRUPT *******************

SIINT:   CLR   SI
         MOV   A,R2                ; LOAD THE OPERATION FIELD
         CJNE  A,#03H,SINTJ1       ; RECEIVE COMMAND PENDING?
         JMP   SIREC               ; YES.
SINTJ1:  CJNE  A,#02H,SINTJ2       ; TRANSMIT-DISCONNECT PENDING?
         JMP   SITDIS              ; YES.
SINTJ2:  JMP   SITRA               ; TRANSMIT COMMAND IS PENDING


;** TRANSMISSION IS DISABLED

SITDIS:  JB    RTS,SINTJ3          ; REQUEST TO SEND ENABLED?
         JNB   TBF,SINTJ3          ; YES. TRANSMISSION DISABLED?
                                   ; YES.
         CLR   P1.2                ; UPDATE STATUS WITH
         SETB  P1.3                ;TRANSMISSION-DISABLED EVENT
         SETB  P1.4                ; (STATUS=D9H)

         CLR   IE0                 ; IGNORE PENDING EXT INT0
         CLR   P1.0
         SETB  P1.0                ; INTERRUPT THE 80186
         JB    P3.2,$              ; WAIT TILL INTERRUPT IS ACKNOWLEDGED
         RETI

;** A FRAME IS TRANSMITTED

SITRA:   JB    RTS,SINTJ3          ; A FRAME TRANSMITTED?
                                   ; YES.
         CLR   P1.2                ; UPDATE STATUS WITH
         SETB  P1.3                ;TRANSMIT-DONE EVENT
         SETB  P1.4                ; (STATUS=C9).

         CLR   IE0
         CLR   P1.0
         SETB  P1.0                ; INTERRUPT THE 80186
         JB    P3.2,$              ; WAIT TILL INTERRUPT IS ACKNOWLEDGED
         RETI                                                    231784-32
; ** A FRAME IS RECEIVED

SIREC:   JB    RBE,SINTJ3          ; RECEIVE BUFFER FULL?
         JNB   BOV,SINTJ4          ; YES. BUFFER OVERRUN?
         SETB  P1.5                ; YES. SET THE E BIT
SINTJ4:  MOV   R0,RFL              ; LOAD R0 WITH RECEIVE BYTE COUNT
         MOV   R1,RBS              ; LOAD R1 WITH RECEIVE BUFFER ADDRESS
         CLR   IE0                 ; IGNORE PENDING EXT INTO (IF ANY)
         SETB  PX0                 ; EXT INT0: PRIORITY 1

         MOV   A,@R1               ; MOVE FIRST BYTE INTO ACC.
         MOVX  @DPTR,A             ; WRITE TO THE COMMAND/DATA REG
         SETB  P1.1                ; ENABLE DMA REQUEST
         INC   R1                  ; INC POINTER TO RECEIVE BUFFER
         JB    P3.3,$              ; WAIT FOR DMA ACK.
         DJNZ  R0,CINTJ7           ; LAST BYTE MOVED?
         SJMP  CINTJ8              ; YES

CINTJ7:  MOV   A,@R1               ; LOAD RECEIVED DATA INTO ACC.
         MOVX  @DPTR,A             ; WRITE TO THE COMMAND/DATA REG.
         INC   R1                  ; INC POINTER TO RECEIVE BUFFER
         JB    P3.3,$              ; WAIT TILL DMA ACK
         DJNZ  R0,CINTJ7           ; LAST BYTE MOVED TO COMMAND/DATA REG?
                                   ; NO. DEPOSIT THE NEXT BYTE
CINTJ8:  MOV   A,RFL               ; LOAD BYTE COUNT
         MOVX  @DPTR,A             ; WRITE TO THE COMMAND/DATA REG
         JB    P3.3,$              ; WAIT FOR DMA ACK.
         MOV   A,STAD              ; LOAD STATION ADDRESS
         MOVX  @DPTR,A             ; WRITE TO THE COMMAND/DATA REG
         JB    P3.3,$              ; WAIT FOR DMA ACK.
         MOV   A,RCB               ; LOAD RECEIVE CONTROL BYTE
         MOVX  @DPTR,A             ; WRITE TO THE COMMAND/DATA REG
         JB    P3.3,$              ; WAIT FOR DMA ACK.
         CLR   P1.1                ; CLEAR DMA REQUEST
         CLR   PX0                 ; EXTERNAL INTERRUPT: PRIORITY 0
                                                                 231784-33
```

Figure A-14. Serial Channel Interrupt Routine

```
                CLR     P1.2            ; UPDATE STATUS WITH
                CLR     P1.3            ;RECEIVE-DONE EVENT
                SETB    P1.4            ; (STATUS=D1H IF E=0)
                CLR     IE0             ; IGNORE PENDING EXT INT0
                CLR     P1.0
                SETB    P1.0            ; INTERRUPT THE 80186
                JB      P3.2,$          ; WAIT TILL INTERRUPT IS ACKNOWLEDGED
                RETI

SINTJ3:         NOP
                RETI
END
```

231784-34

Figure A-14. Serial Channel Interrupt Routine (Continued)

```
                        CLR   SI.2          ; UPDATE STATUS WORD
                        CLR   TI.2          ; RECORD I-CODE EVENT
                        SETB  TI.1          ; (START-SIO IF NEC)
                        CLR   ITO           ; CHOOSE SENDING VS. INTO
                        CLR   PI.0
                        SETB  PI.0          ; INTERRUPT THE 8096
                        JB    P0.3,$        ; WAIT UNTIL BUSREQUEST IS ACKNOWLEDGED
                        RETI

BIRTH:                  NOP
                        NOP
END:
```

RUPI™-44 Family

19

THE RUPI™-44 FAMILY: MICROCONTROLLER WITH ON-CHIP COMMUNICATION CONTROLLER

INTRODUCTION

The RUPI-44 family is designed for applications requiring local intelligence at remote nodes, and communication capability among these distributed nodes. The RUPI-44 integrates onto a single chip Intel's highest performance microcontroller, the 8051-core, with an intelligent and high performance Serial communication controller, called the Serial Interface Unit, or SIU. See Figure 1. This dual controller architecture allows complex control and high speed data communication functions to be realized cost effectively.

The RUPI-44 family consists of three pin compatible parts:

- 8344—8051 Microcontroller with SIU
- 8044—An 8344 with 4K bytes of on-chip ROM program memory
- 8744—An 8344 with 4K bytes of on-chip EPROM program memory

1.0 ARCHITECTURE OVERVIEW

The 8044's dual controller architecture enables the RUPI to perform complex control tasks and high speed communication in a distributed network environment.

The 8044 microcontroller is the 8051-core, and maintains complete software compatibility with it. The microcontroller contains a powerful CPU with on-chip peripherals, making it capable of serving sophisticated real-time control applications such as instrumentation, industrial control, and intelligent computer peripherals. The microcontroller features on-chip peripherals such as two 16-bit timer/counters and 5 source interrupt capability with programmable priority levels. The microcontroller's high performance CPU executes most instructions in 1 microsecond, and can perform an 8×8 multiply in 4 microseconds. The CPU features a Boolean processor that can perform operations on 256 directly addressable bits. 192 bytes of on-chip data RAM can be extended to 64K bytes externally. 4K bytes of on-chip program ROM can be extended to 64K bytes externally. The CPU and SIU run concurrently. See Figure 2.

The SIU is designed to perform serial communications with little or no CPU involvement. The SIU supports data rates up to 2.4 Mbps, externally clocked, and 375 Kbps self clocked (i.e., the data clock is recovered by an on-chip digital phase locked loop). SIU hardware supports the HDLC/SDLC protocol: zero bit insertion/deletion, address recognition, cyclic redundancy check, and frame number sequence check are automatically performed.

The SIU's Auto mode greatly reduces communication software overhead. The AUTO mode supports the SDLC Normal Response Mode, by performing secondary station responses in hardware without any CPU involvement. The Auto mode's interrupt control and frame sequence numbering capability eliminates software overhead normally required in conventional systems. By using the Auto mode, the CPU is free to concentrate on real time control of the application.

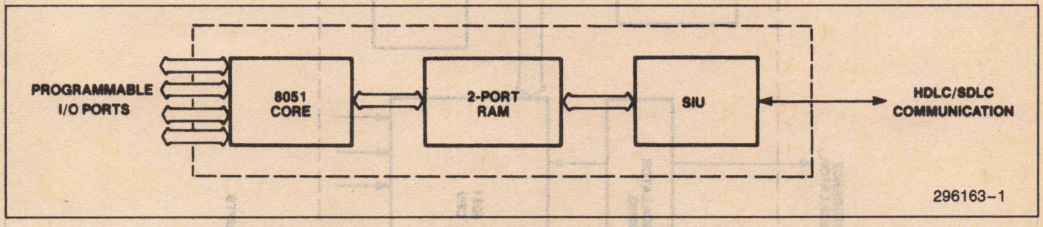


Figure 1. RUPI™-44 Dual Controller Architecture

Order Number: 296163-001

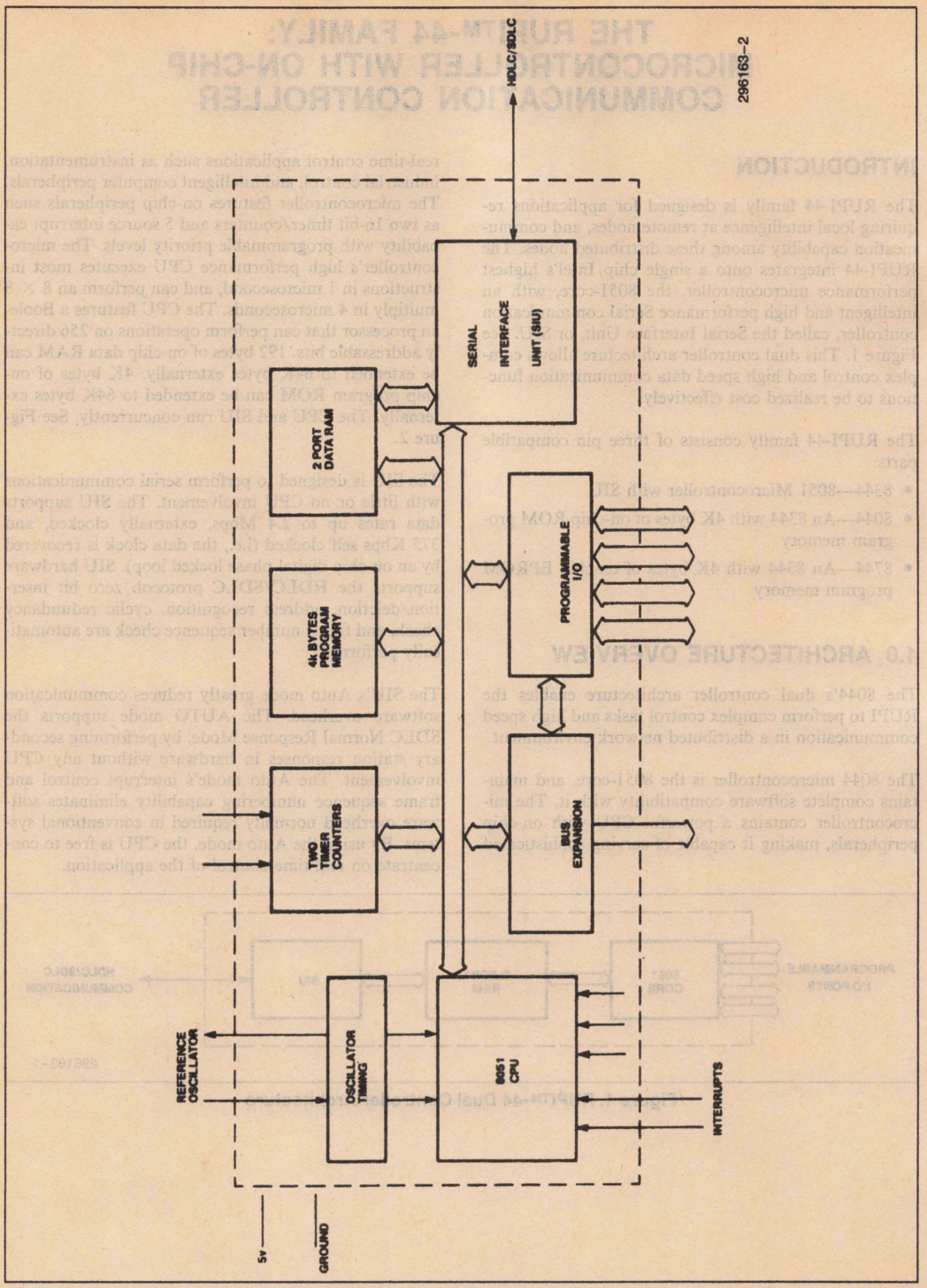


Figure 2. Simplified 8044 Block Diagram

2.0 THE HDLC/SDLC PROTOCOLS

2.1 HDLC/SDLC Advantages over Async

The High Level Data Link Control, HDLC, is a standard communication link control established by the International Standards Organization (ISO). SDLC is a subset of HDLC.

HDLC and SDLC are both well recognized standard serial protocols. The Synchronous Data Link Control, SDLC, is an IBM standard communication protocol. IBM originally developed SDLC to provide efficient, reliable and simple communication between terminals and computers.

The major advantages of SDLC/HDLC over Asynchronous communications protocol (Async):

- SIMPLE: Data Transparency
- EFFICIENT: Well Defined Message-Level Operation
- RELIABLE: Frame Check Sequence and Frame Numbering

The SDLC reduces system complexity. HDLC/SDLC are "data transparent" protocols. Data transparency means that an arbitrary data stream can be sent without concern that some of the data could be mistaken for a protocol controller. Data transparency relieves the communication controller having to detect special characters.

SDLC/HDLC provides more data throughout than Async. SDLC/HDLC runs at Message-level Operation which transmits multiple bytes within the frame, whereas Async is based on character-level operation. Async transmits or receives a character at a time. Since Async requires start and stop bits in every transmission, there is a considerable waste of overhead compared to SDLC/HDLC.

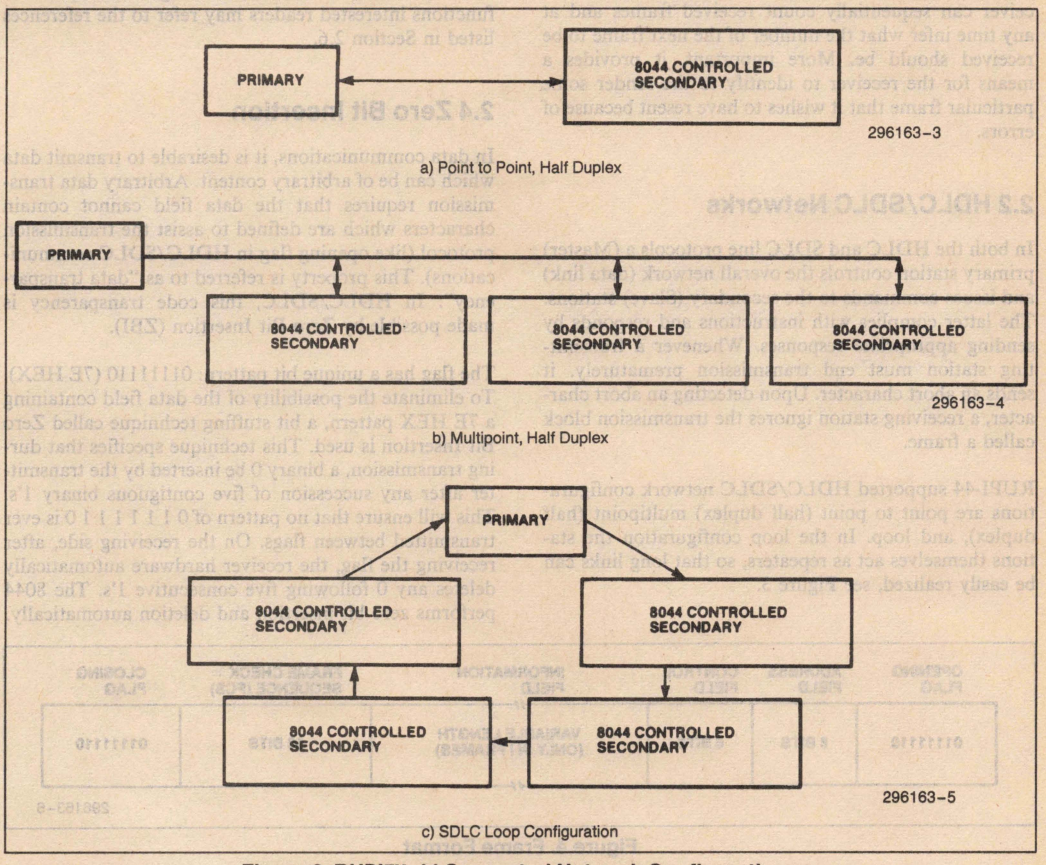


Figure 3. RUPI™-44 Supported Network Configurations

Due to SDLC/HDLC's well delineated field (see Figure 4) the CPU does not have to interpret character by character to determine control field and information field. In the case of Async, CPU must look at each character to interpret what it means. The practical advantage of such feature is straight forward use of DMA for information transfer.

In addition, SDLC/HDLC further improves Data throughput using implied Acknowledgement of transferred information. A station using SDLC/HDLC may acknowledge previously received information while transmitting different information in the same frame. In addition, up to 7 messages may be outstanding before an acknowledgement is required.

The HDLC/SDLC protocol can be used to realize reliable data links. Reliable Data transmission is ensured at the bit level by sending a frame check sequence, cyclic redudancy checking, within the frame. Reliable frame transmission is ensured by sending a frame number identification with each frame. This means that a receiver can sequentially count received frames and at any time infer what the number of the next frame to be received should be. More important, it provides a means for the receiver to identify to the sender some particular frame that it wishes to have resent because of errors.

2.2 HDLC/SDLC Networks

In both the HDLC and SDLC line protocols a (Master) primary station controls the overall network (data link) and issues commands to the secondary (Slave) stations. The latter complies with instructions and responds by sending appropriate responses. Whenever a transmitting station must end transmission prematurely, it sends an abort character. Upon detecting an abort character, a receiving station ignores the transmission block called a frame.

RUPI-44 supported HDLC/SDLC network configurations are point to point (half duplex) multipoint (half duplex), and loop. In the loop configuration the stations themselves act as repeaters, so that long links can be easily realized, see Figure 3.

2.3 Frames

An HDLC/SDLC frame consists of five basic fields: Flag, Address, Control, Data and Error Detection. A frame is bounded by flags—opening and closing flags. An address field is 8 bits wide in SDLC, extendable to 2 or more bytes in HDLC. The control field is also 8 bits wide, extendable to two bytes in HDLC. The SDLC data field or information field may be any number of bytes. The HDLC data field may or may not be on an 8 bit boundary. A powerful error detection code called Frame Check Sequence contains the calculated CRC (Cycle Redundancy Code) for all the bits between the flags. See Figure 4.

In HDLC and SDLC are three types of frames; an Information Frame is used to transfer data, a Supervisory Frame is used for control purposes, and a Nonsequenced Frame is used for initialization and control of the secondary stations.

For a more detailed discussion of higher level protocol functions interested readers may refer to the references listed in Section 2.6.

2.4 Zero Bit Insertion

In data communications, it is desirable to transmit data which can be of arbitrary content. Arbitrary data transmission requires that the data field cannot contain characters which are defined to assist the transmission protocol (like opening flag in HDLC/SDLC communications). This property is referred to as "data transparency". In HDLC/SDLC, this code transparency is made possible by Zero Bit Insertion (ZBI).

The flag has a unique bit pattern: 01111110 (7E HEX). To eliminate the possibility of the data field containing a 7E HEX pattern, a bit stuffing technique called Zero Bit Insertion is used. This technique specifies that during transmission, a binary 0 be inserted by the transmitter after any succession of five contiguous binary 1's. This will ensure that no pattern of 0 1 1 1 1 1 1 0 is ever transmitted between flags. On the receiving side, after receiving the flag, the receiver hardware automatically deletes any 0 following five consecutive 1's. The 8044 performs zero bit insertion and deletion automatically.

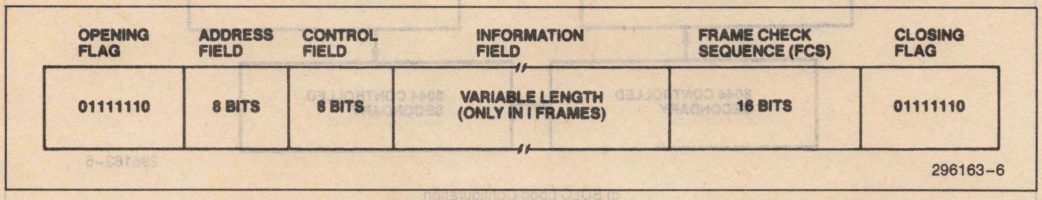


Figure 4. Frame Format

2.5 Non-return to Zero Inverted (NR21)

NRZI is a method of clock and data encoding that is well suited to the HDLC/SDLC protocol. It allows HDLC/SDLC protocols to be used with low cost asynchronous modems. NRZI coding is done at the transmitter to enable clock recovery from the data at the receiver terminal by using standard digital phase locked loop (DPLL) techniques. NRZI coding specifies that the signal condition does not change for transmitting a 1, while a 0 causes a change of state. NRZI coding ensures that an active data line will have a transition at least every 5-bit times (recall Zero Bit Insertion), while contiguous 0's will cause a change of state. Thus, ZBI and NRZI encoding makes it possible for the 8044's on-chip DPLL to recover a receive clock (from received data) synchronized to the received data and at the same time ensure data transparency.

2.6 References

1. *IBM Synchronous Data Link Control General Information GA27-3093-2 File No. GENL-09.*
2. *Standard Network Access Protocol Specification, DATAPAC Trans-Canada Telephone System CCG111.*
3. *IBM 3650 Retail Store System Loop Interface OEM Information, IBM, GA27-3098-0.*
4. *Guidebook to Data Communications, Training Manual, Hewlett-Packard 5955-1715.*
5. *"Serial Backplane Suits Multiprocessor Architectures", Mike Webb, Computer Design, July 1984, pp. 85–96.*
6. *"Serial Bus Simplifies Distributed Control", P.D. MacWilliams, Control Engineering, June 1984, pp. 101–104.*
7. *"Chips Support Two Local Area Networks", Bob Dahlberg, Computer Design, May 1984, pp. 107–114.*
8. *"Build a VLSI-based Workstation for the Ethernet Environment", Mike Webb, EDN, 23 February 1984, pp. 297–307.*
9. *"Networking With the 8044", Young Sohn & Charles Gopen, Digital Design, May 1984, pp. 136–137.*

3.0 RUPI™-44 DESIGN SUPPORT

3.1 Design Tool Support

A critical design consideration is time to market. Intel provides a sophisticated set of design tools to speed hardware and software development time of 8044 based products. These include ICE-44, ASM-51, PL/M-51, and EMV-44.

Figure 5. RUPI™-44 Development Support Configuration Intellec® System, ICE™-44 Buffer Box, and ICE-44 Module Plugged into a User Prototype Board

A primary tool is the 8044 In Circuit Emulator, called ICE-44. See Figure 5. In conjunction with Intel's Intellec® Microprocessor Development System, the ICE-44 emulator allows hardware and software development to proceed interactively. This approach is more effective than the traditional method of independent hardware and software development followed by system integration. With the ICE-44 module, prototype hardware can be added to the system as it is designed. Software and hardware integration occurs while the product is being developed.

The ICE-44 emulator assists four stages of development:

1) Software Debugging

   It can be operated without being connected to the user's system before any of the user's hardware is available. In this stage ICE-44 debugging capabilities can be used in conjunction with the Intellec text editor and 8044 macroassembler to facilitate program development.

2) Hardware Development

   The ICE-44 module's precise emulation characteristics and full-speed program RAM make it a valuable tool for debugging hardware, including the time-critical SDLC serial port, parallel port, and timer interfaces.

3) System Integration

Integration of software and hardware can begin when any functional element of the user system hardware is connected to the 8044 socket. As each section of the user's hardware is completed, it is added to the prototype. Thus, each section of the hardware and software is system tested in real-time operation as it becomes available.

4) System Test

When the user's prototype is complete, it is tested with the final version of the user system software. The ICE-44 module is then used for real-time emulation of the 8044 to debug the system as a completed unit.

The final product verification test may be performed using the 8744 EPROM version of the 8044 microcomputer. Thus, the ICE-44 module provides the user with the ability to debug a prototype or production system at any stage in its development.

A conversion kit, ICE-44 CON, is available to upgrade an ICE-51 module to ICE-44.

Intel's ASM-51 Assembler supports the 8044 special function registers and assembly program development. PL/M-51 provides designers with a high level language for the 8044. Programming in PL/M can greatly reduce development time, and ensure quick time to market.

These tools have recently been expanded with the addition of the EMV-44CON. This conversion kit allows you to convert an EMV-51 into an EMV-44 emulation vehicle. The resultant low cost emulator is designed for use with an iPDS Personal Development System, which also supports the ASM-51 assembler and PL/M-51. See Figure 6.

Emulation support is similar to the ICE-44 with support for Software and Hardware Development, System Integration, and System Test. The iPDS's rugged portability and ease of use also make it an ideal system for production tests and field service of your finished design. In addition, the iPDS offers EPROM programming module for the 8744, and direct communications with the 8044-based BITBUS via an optional iSBX-344 distributed control module.

Figure 6. RUPI-44 iPDS Personal Development System, EMV-44 Buffer Box, and EMV-44 Module Plugged into a User Prototype Board

3.2 8051 Workshop

Intel provides 8051 training to its customers through the 5-day 8051 workshop. Familiarity with the 8051 and 8044 is achieved through a combination of lecture and laboratory exercises.

For designers not familiar with the 8051, the workshop is an effective way to become proficient with the 8051 architecture and capabilities.

8044 Architecture

20

8044 ARCHITECTURE

GENERAL

The 8044 is based on the 8051 core. The 8044 replaces the 8051's serial port with an intelligent HDLC/SDLC controller called the Serial Interface or SIU. Thus the differences between the two result from the 8044's increased on-chip RAM (192 bytes) and additional special function registers necessary to control the SIU. Aside from the increased memory, the SIU itself, and differences in 5 pins (for the serial port), the 8044 and 8051 are compatible.

This chapter describes the differences between the 8044 and 8051. Information pertaining to the 8051 core, eg. instruction set, port operation, EPROM programming, etc. is located in the 8051 sections of this manual.

A block diagram of the 8044 is shown in Figure 1. The pinpoint is shown on the inside front cover.

1.0 MEMORY ORGANIZATION OVERVIEW

The 8044 maintains separate address spaces for Program Memory and Data Memory. The Program Memory can be up to 64K bytes long, of which the lowest 4K bytes are in the on-chip ROM.

If the $\overline{EA}$ pin is held high, the 8044 executes out of internal ROM unlwess the Program Counter exceeds 0FFFH. Fetches from locations 1000H through FFFFH are directed to external Program Memory.

If the $\overline{EA}$ pin is held low, the 8044 fetches all instructions from external Program Memory.

The Data Memory consists of 192 bytes of on-chip RAM, plus 35 Special Function Registers, in addition to which the device is capable of accessing up to 64K bytes of external data memory.

The Program Memory uses 16-bit addresses. The external Data Memory can use erither 8-bit or 16-bit addresses. The internal Data Memory uses 8-bit addresses, which provide a 256-location address space. The lower 192 addresses access the on-chip RAM. The Special Function Registers occupy various locations in the upper 128 bytes of the same address space.

The lowest 32 bytes in the internal RAM (locations 00 through 1FH) are divided into 4 banks of registers, each bank consisting of 8 bytes. Any one of these banks can be selected to be the "working registers" of the CPU, and can be accessed by a 3-bit address in the same byte as the opcode of an instruction. Thus, a large number of instructions are one-byte instructions.

The next higher 16 bytes of the internal RAM (locations 20H through 2FH) have individually addressable bits. These are provided for use as software flags or for one-bit (Boolean) processing. This bit-addressing capability is an important feature of the 8044. In addition to the 128 individually addressable bits in RAM, twelve of the Special Function Registers also have individually addressable bits.

A memory map is shown in Figure 2.

1.1 Special Function Registers

The Special Function Registers are as follows:

*	ACC	Accumulator (A Register)
*	B	B Register
*	PSW	Program Status Word
	SP	Stack Pointer
	DPTR	Data Pointer (consisting of DPH AND DPL)
*	P0	Port 0
*	P1	Port 1
*	P2	Port 2
*	P3	Port 3
*	IP	Interrupt Priority
*	IE	Interrupt Enable
	TMOD	Timer/Counter Mode
*	TCON	Timer/Counter Control
	TH0	Timer/Counter 0 (high byte)
	TL0	Timer/Counter 0 (low byte)
	TH1	Timer/Counter 1 (high byte)
	TL1	Timer/Counter 1 (low byte)
	SMD	Serial Mode
*	STS	Status/Command
*	NSNR	Send/Receive Count
	STAD	Station Address
	TBS	Transmit Buffer Start Address
	TBL	Transmit Buffer Length
	TCB	Transmit Control Byte
	RBS	Receive Buffer Start Address
	RBL	Receive Buffer Length
	RFL	Received Field Length
	RCB	Received Control Byte
	DMA CNT	DMA Count
	FIFO	FIFO (three bytes)
	SIUST	SIU State Counter
	PCON	Power Control

The registers marked with * are both byte- and bit-addressable.

Order Number: 296164-001

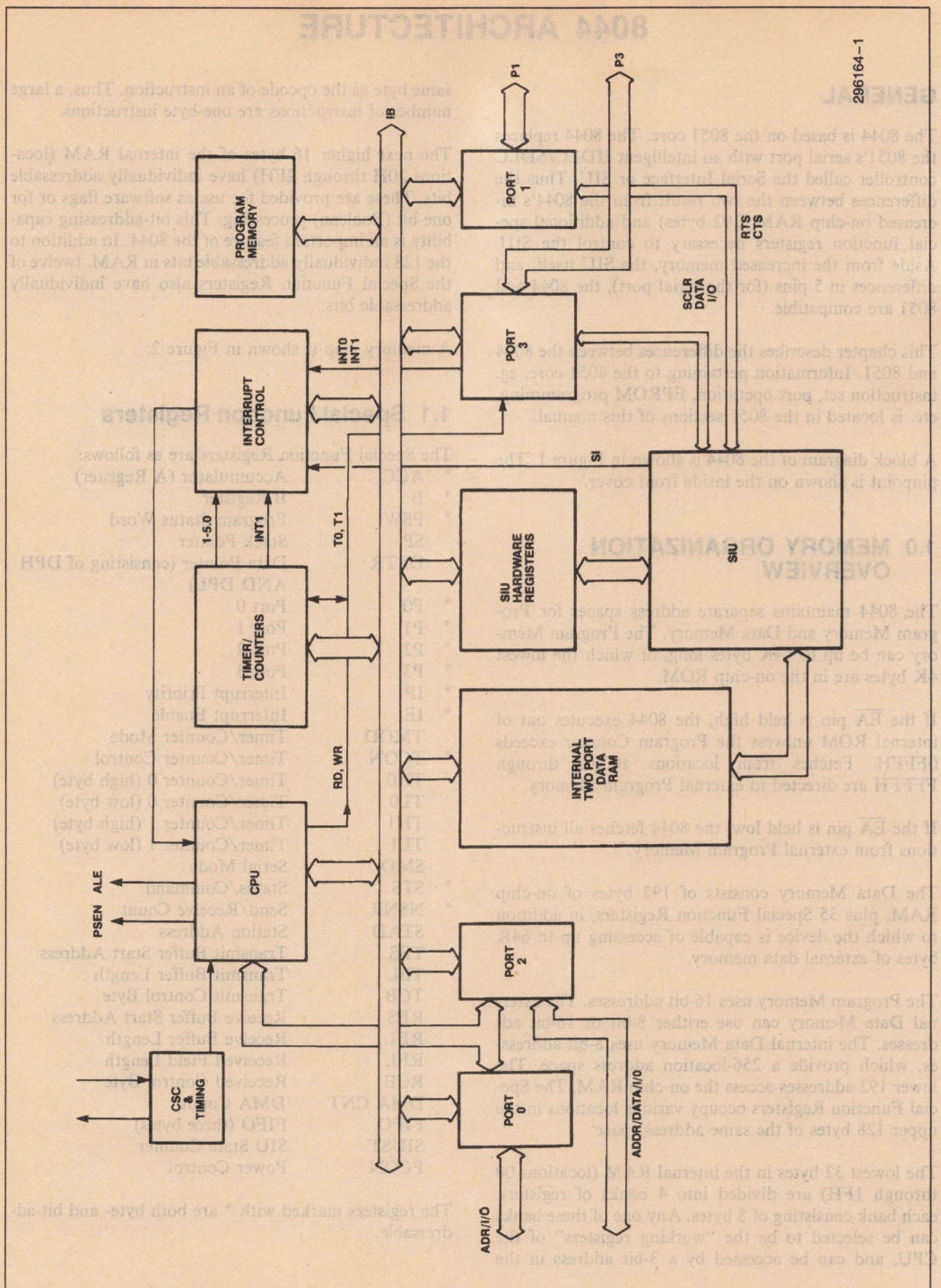


Figure 1. RUPI™ Block Diagram

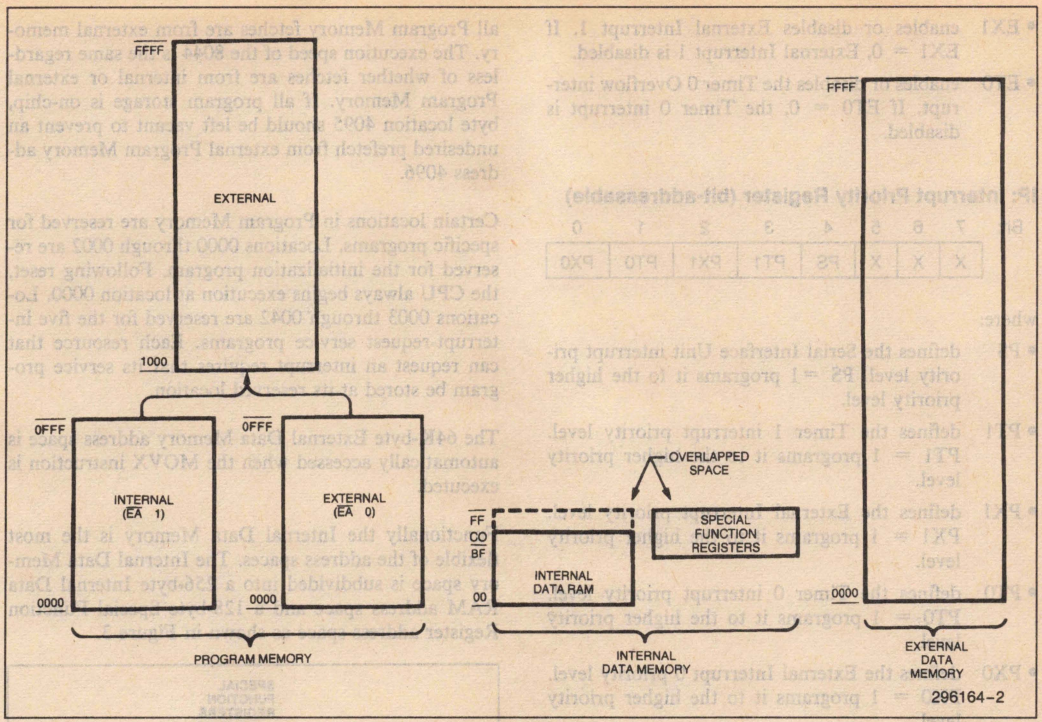


Figure 2. RUPI™-44 Memory Map

Stack Pointer

The Stack Pointer is 8 bits wide. The stack can reside anywhere in the 192 bytes of on-chip RAM. When the 8044 is reset, the stack pointer is initialized to 07H. When executing a PUSH or a CALL, the stack pointer is incremented before data is stored, so the stack would begin at location 08H.

1.2 Interrupt Control Registers

The Interrupt Request Flags are as listed below:

Source	*Request Flag*	*Location*
External Interrupt 0	$\overline{INT0}$, if IT0 = 0 IE0, if IT0 = 1	P3.2 TCON.1
Timer 0 Overflow	TF0	TCON.5
External Interrupt 1	$\overline{INT1}$, if IT1 = 0 IE1, if IT1 = 1	P3.3 TCON.3
Timer 1 Overflow	TF1	TCON.7
Serial Interface Unit	SI	STS.4

External Interrupt control bits IT0 and IT1 are in TCON.0 and TCON.2, respectively. Reset leaves all flags inactive, with IT0 and IT1 cleared.

All the interrupt flags can be set or cleared by software, with the same effect as by hardware.

The Enable and Priority Control Registers are shown below. All of these control bits are set or cleared by software. All are cleared by reset.

IE: Interrupt Enable Register (bit-addressable)

Bit:	7	6	5	4	3	2	1	0
	EA	X	X	ES	ET1	EX1	ET0	EX0

where:

- EA disables all interrupts. If EA = 0, no interrupt will be acknowledged. If EA = 1, each interrupt source is individually enabled or disabled by setting or clearing its enable bit.
- ES enables or disables the Serial Interface Unit interrupt. If ES = 0, the Serial Interface Unit interrupt is disabled.
- ET1 enables or disables the Timer 1 Overflow interrupt. If ET1 = 0, the Timer 1 interrupt is disabled.

- EX1 enables or disables External Interrupt 1. If EX1 = 0, External Interrupt 1 is disabled.
- ET0 enables or disables the Timer 0 Overflow interrupt. If ET0 = 0, the Timer 0 interrupt is disabled.

IP: Interrupt Priority Register (bit-addressable)

Bit:	7	6	5	4	3	2	1	0
	X	X	X	PS	PT1	PX1	PT0	PX0

where:

- PS defines the Serial Interface Unit interrupt priority level. PS = 1 programs it to the higher priority level.
- PT1 defines the Timer 1 interrupt priority level. PT1 = 1 programs it to the higher priority level.
- PX1 defines the External Interrupt priority level. PX1 = 1 programs it to the higher priority level.
- PT0 defines the Timer 0 interrupt priority level. PT0 = 1 programs it to the higher priority level.
- PX0 defines the External Interrupt 0 priority level. PX0 = 1 programs it to the higher priority level.

2.0 MEMORY ORGANIZATION DETAILS

In the 8044 family the memory is organized over three address spaces and the program counter. The memory spaces shown in Figure 2 are the:

- 64K-byte Program Memory address space
- 64K-byte External Data Memory address space
- 320-byte Internal Data Memory address space

The 16-bit Program Counter register provides the 8044 with its 64K addressing capabilities. The Program Counter allows the user to execute calls and branches to any location within the Program Memory space. There are no instructions that permit program execution to move from the Program Memory space to any of the data memory spaces.

In the 8044 and 8744 the lower 4K of the 64K Program Memory address space is filled by internal ROM and EPROM, respectively. By tying the $\overline{EA}$ pin high, the processor can be forced to fetch from the internal ROM/EPROM for Program Memory addresses 0 through 4K. Bus expansion for accessing Program Memory beyond 4K is automatic since external instruction fetches occur automatically when the Program Counter increases above 4095. If the $\overline{EA}$ pin is tied low all Program Memory fetches are from external memory. The execution speed of the 8044 is the same regardless of whether fetches are from internal or external Program Memory. If all program storage is on-chip, byte location 4095 should be left vacant to prevent an undesired prefetch from external Program Memory address 4096.

Certain locations in Program Memory are reserved for specific programs. Locations 0000 through 0002 are reserved for the initialization program. Following reset, the CPU always begins execution at location 0000. Locations 0003 through 0042 are reserved for the five interrupt-request service programs. Each resource that can request an interrupt requires that its service program be stored at its reserved location.

The 64K-byte External Data Memory address space is automatically accessed when the MOVX instruction is executed.

Functionally the Internal Data Memory is the most flexible of the address spaces. The Internal Data Memory space is subdivided into a 256-byte Internal Data RAM address space and a 128-byte Special Function Register address space as shown in Figure 3.

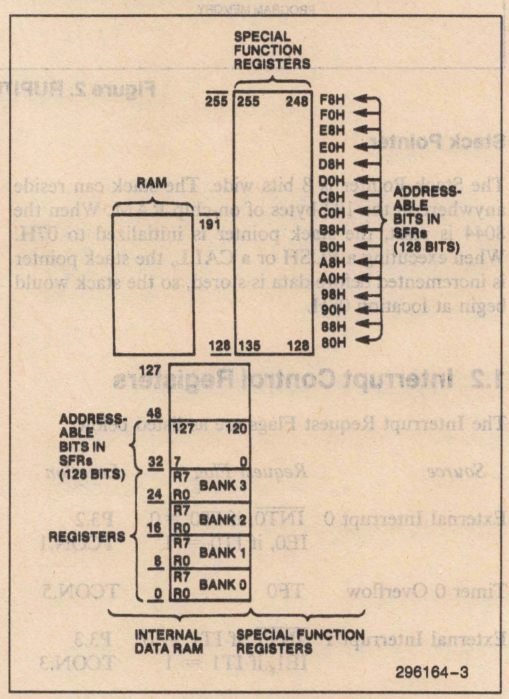


Figure 3. Internal Data Memory Address Space

The Internal Data RAM address space is 0 to 255. Four 8-Register Banks occupy locations 0 through 31. The stack can be located anywhere in the Internal Data RAM address space. In addition, 128 bit locations of the on-chip RAM are accessible through Direct Addressing. These bits reside in Internal Data RAM at byte locations 32 through 47. Currently locations 0 through 191 of the Internal Data RAM address space are filled with on-chip RAM.

The stack depth is limited only by the available Internal Data RAM, thanks to an 8-bit reloadable Stack Pointer. The stack is used for storing the Program Counter during subroutine calls and may be used for passing parameters. Any byte of Internal Data RAM or Special Function Register accessible though Direct Addressing can be pushed/popped.

The Special Function Register address space is 128 to 255. All registers except the Program Counter and the four 8-Register Banks reside here. Memory mapping the Special Function Registers allows them to be accessed as easily as internal RAM. As such, they can be operated on by most instructions. In the overlapping memory space (address 128-191), indirect addressing is used to access RAM, and direct addressing is used to access the SFR's. The SFR's at addresses 192–255 are also accessed using direct addressing. The Special Function Registers are listed in Figure 4. Their mapping in the Special Function Register address space is shown in Figures 5 and 6.

Performing a read from a location of the Internal Data memory where neither a byte of Internal Data RAM (i.e., RAM addresses 192-255) nor a Special Function Register exists will access data of indeterminable value.

Architecturally, each memory space is a linear sequence of 8-bit wide bytes. By Intel convention the storage of multi-byte address and data operands in program and data memories is the least significant byte at the low-order address and the most significant byte at the high-order address. Within byte X, the most significant bit is represented by X.7 while the least significant bit is X.0. Any deviation from these conventions will be explicitly stated in the text.

2.1 Operand Addressing

There are five methods of addressing source operands. They are Register Addressing, Direct Addressing, Register-Indirect Addressing, Immediate Addressing

ARITHMETIC REGISTERS:
Accumulator*, B register*,
Program Status Word*
POINTERS:
Stack Pointer, Data Pointer (high & low)
PARALLEL I/O PORTS:
Port 3*, Port 2*, Port 1*, Port 0*
INTERRUPT SYSTEM:
Interrupt Priority Control*,
Interrupt Enable Control*
TIMERS:
Timer Mode, Timer Control*, Timer 1 (high & low), Timer 0 (high & low)
SERIAL INTERFACE UNIT:
Transmit Buffer Start,
Transmit Buffer Length,
Transmit Control Byte,
Send Count Receive Count*,
DMA Count,
Station Address
Receive Field Length
Receive Buffer Start
Receive Buffer Length
Receive Control Byte,
Serial Mode,
Status Register.*

***Bits in these registers are bit addressable.**

Figure 4. Special Function Registers

ARITHMETIC REGISTERS:
Accumulator*, B register*,
Program Status Word*
POINTERS:
Stack Pointer, Data Pointer (high & low)
PARALLEL I/O PORTS:
Port 3*, Port 2*, Port 1*, Port 0*
INTERRUPT SYSTEM:
Interrupt Priority Control*,
Interrupt Enable Control*
TIMERS:
Timer Mode, Timer Control*, Timer 1 (high & low), Timer 0 (high & low)
SERIAL INTERFACE UNIT:
Serial Mode, Status/Command*,
Send/Receive Count*, Station Address,
Transmit Buffer Start Address,
Transmit Buffer Length,
Transmit Control Byte,
Receive Buffer Start Address,
Receive Buffer Length,
Receive Field Length,
Receive Control Byte,
DMA Count,
FIFO (three bytes),
SIU Controller State Counter

***Bits in these registers are bit-addressable.**

Figure 5. Mapping of Special Function Registers

and Base-Register-plus Index-Register-Indirect Addressing. The first three of these methods can also be used to address a destination operand. Since operations in the 8044 require 0 (NOP only), 1, 2, 3 or 4 operands, these five addressing methods are used in combinations to provide the 8044 with its 21 addressing modes.

Most instructions have a "destination, source" field that specifies the data type, addressing methods and operands involved. For operations other than moves, the destination operand is also a source operand. For example, in "subtract-with-borrow A,#5" the A register receives the result of the value in register A minus 5, minus C.

Most operations involve operands that are located in Internal Data Memory. The selection of the Program Memory space or External Data Memory space for a second operand is determined by the operation mnemonic unless it is an immediate operand. The subset of the Internal Data Memory being addressed is determined by the addressing method and address value. For example, the Special Function Registers can be accessed only through Direct Addressing with an address of 128-255. A summary of the operand addressing methods is shown in Figure 6. The following paragraphs describe the five addressing methods.

2.2 Register Addressing

Register Addressing permits access to the eight registers (R7-R0) of the selected Register Bank (RB). One of the four 8-Register Banks is selected by a two-bit field in the PSW. The registers may also be accessed through Direct Addressing and Register-Indirect Addressing, since the four Register Banks are mapped into the lowest 32 bytes of internal Data RAM as shown in Figures 9 and 10. Other Internal Data Memory locations that are addressed as registers are A, B, C, AB and DPTR.

2.3 Direct Addressing

Direct Addressing provides the only means of accessing the memory-mapped byte-wide Special Function Registers and memory mapped bits within the Special Function Registers and Internal Data RAM. Direct Addressing of bytes may also be used to access the lower 128 bytes of Internal Data RAM. Direct Addressing of bits gains access to a 128 bit subset of the Special Function Registers as shown in Figures 5, 6, 9, and 10.

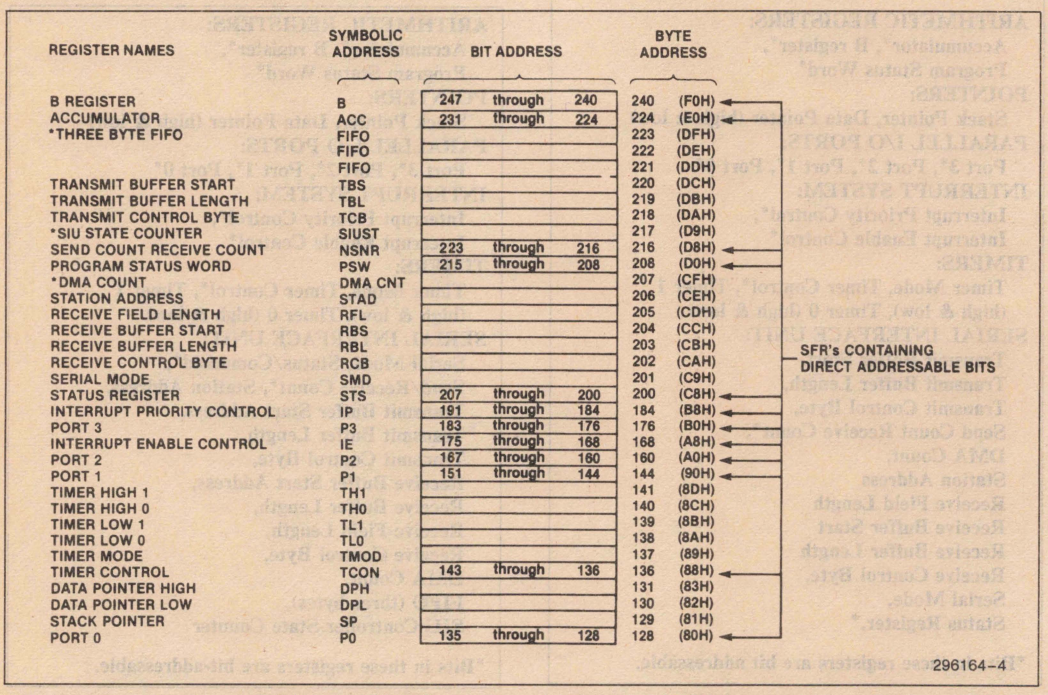


Figure 6. Mapping of Special Function Registers

Direct Byte Address	Bit Address (MSB)							(LSB)	Hardware Register Symbol
240	F7	F6	F5	F4	F3	F2	F1	F0	B
224	E7	E6	E5	E4	E3	E2	E1	E0	ACC
	NS2	NS1	NS0	SES	NR2	NR1	NR0	SER	
216	DF	DE	DD	DC	DB	DA	D9	D8	NSNR
	CY	AC	FO	RS1	RSO	OV		P	
204	D7	D6	D5	D4	D3	D2	D1	D0	PSW
	TBF	RE	RTS	SI	BV	CPB	AM	RBP	
200	CF	CE	CD	CC	CB	CA	C9	C8	STS
				PS	PT1	PX1	PT0	PX0	
184	—	—	—	BC	BB	BA	B9	B8	1P
176	B7	B6	B5	B4	B3	B2	B1	B0	P3
	EA			E5	ET1	EX1	ET0	EX0	
168	AF	—	—	AC	AB	AA	A9	A8	1E
160	A7	A6	A5	A4	A3	A2	A1	A0	P2
144	97	96	95	94	93	92	91	90	P1
	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	
136	8F	8E	8D	8C	8B	8A	89	88	TCON
128	87	86	85	84	83	82	81	80	P0

Figure 7. Special Function Register Bit Address

- **Register Addressing**
  - — **R7-R0**
  - — **A, B, C (bit), AB (two bytes), DPTR (double byte)**
- **Direct Addressing**
  - — **Lower 128 bytes of Internal Data RAM**
  - — **Special Function Registers**
  - — **128 bits in subset of Special Function Register address space**
- **Register-Indirect Addressing**
  - — **Internal Data RAM [@R1, @R0, @SP (PUSH and POP only)]**
  - — **Least Significant Nibbles in Internal Data RAM (@R1, @R0)**
  - — **External Data Memory (@R1, @R0, @DPTR)**
- **Immediate Addressing**
  - — **Program Memory (in-code constant)**
- **Base-Register-plus Index-Register-Indirect Addressing**
  - — **Program Memory (@ DPTR + A, @ PC + A)**

Figure 8. Operand Addressing Methods

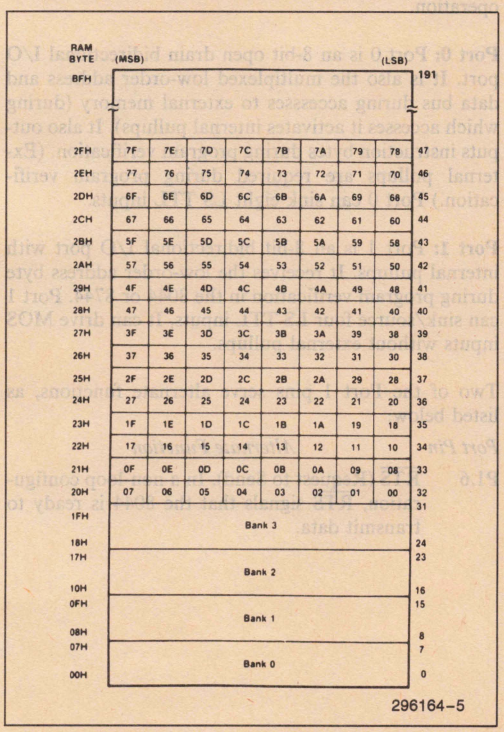


Figure 9. RAM Bit Address

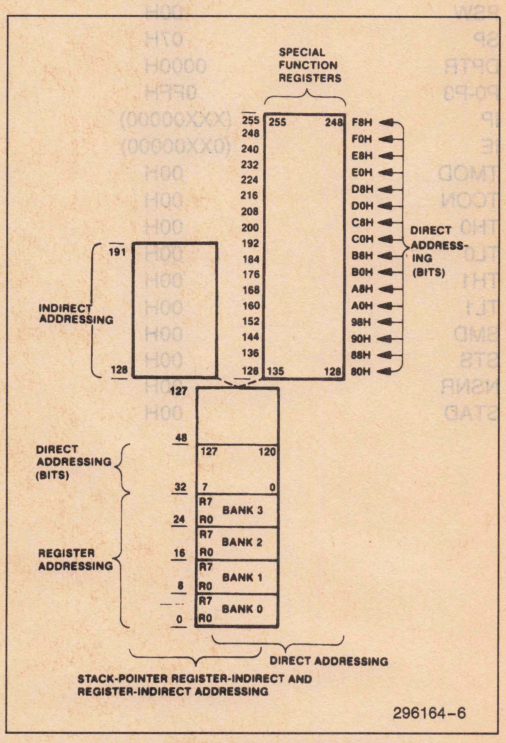


Figure 10. Addressing Operands in Internal Data Memory

Register-Indirect Addressing using the content of R1 or R0 in the selected Register Bank, or using the content of the Stack Pointer (PUSH and POP only), addresses the Internal Data RAM. Register-Indirect Addressing is also used for accessing the External Data Memory. In this case, either R1 or R0 in the selected Register Bank may be used for accessing locations within a 256-byte block. The block number can be preselected by the contents of a port. The 16-bit Data Pointer may be used for accessing any location within the full 64K external address space.

3.0 RESET

Reset is accomplished by holding the RST pin high for at least two machine cycles (24 oscillator periods) *while the oscillator is running.* The CPU responds by executing an internal reset. It also configures the ALE and PSEN pins as inputs. (They are quasi-bidirectional.) The internal reset is executed during the second cycle in which RST is high and is repeated every cycle until RST goes low. It leaves the internal registers as follows:

Register	Content
PC	0000H
A	00H
B	00H
PSW	00H
SP	07H
DPTR	0000H
P0-P3	0FFH
IP	(XXX00000)
IE	(0XX00000)
TMOD	00H
TCON	00H
TH0	00H
TL0	00H
TH1	00H
TL1	00H
SMD	00H
STS	00H
NSNR	00H
STAD	00H
TBS	00H
TBL	00H
TCB	00H
RBS	00H
RBL	00H
RFL	00H
RCB	00H
DMA CNT	00H
FIFO1	00H
FIFO2	00H
FIFO3	00H
SIUST	01H
PCON	(0XXXXXXX)

The internal RAM is not affected by reset. When VCC is turned on, the RAM content is indeterminate unless VPD was applied prior to VCC being turned off (see Power Down Operation.)

4.0 RUPI™-44 FAMILY PIN DESCRIPTION

VSS: Circuit ground potential.

VCC: Supply voltage during programming (of the 8744), verification (of the 8044 or 8744), and normal operation.

Port 0: Port 0 is an 8-bit open drain bidirectional I/O port. It is also the multiplexed low-order address and data bus during accessses to external memory (during which accesses it activates internal pullups). It also outputs instruction bytes during program verification. (External pullups are required during program verification.) Port 0 can sink eight LS TTL inputs.

Port 1: Port 1 is an 8-bit bidirectional I/O port with internal pullups. It receives the low-order address byte during program verification in the 8044 or 8744. Port 1 can sink/source four LS TTL inputs, It can drive MOS inputs without external pullups.

Two of the Port 1 pins serve alternate functions, as listed below:

Port Pin	*Alternate Function*
P1.6	$\overline{RTS}$ (Request to Send). In a non-loop configuration, $\overline{RTS}$ signals that the 8044 is ready to transmit data.

8044 Serial Interface

21

THE RUPI™-44 SERIAL INTERFACE UNIT

SERIAL INTERFACE

The serial interface provides a high-performance communication link. The protocol used for this communication is based on the IBM Synchronous Data Link Control (SDLC). The serial interface also supports a subset of the ISO HDLC (International Standards Organization High-Level Data Link Control) protocol.

The SDLC/HDLC protocols have been accepted as standard protocols for many high-level teleprocessing systems. The serial interface performs many of the functions required to service the data link without intervention from the 8044's own CPU. The programmer is free to concentrate on the 8044's function as a peripheral controller, rather than having to deal with the details of the communication process.

Five pins on the 8044 are involved with the serial interface:

Pin 7	$\overline{RTS}$/P16
Pin 8	$\overline{CTS}$/P17
Pin 10	I/$\overline{O}$/RXD/P30
Pin 11	DATA/TXD/P31
Pin 15	SCLK/T1//P35

Figure 1 is a functional block diagram of the serial interface unit (SIU). More details on the SIU hardware are given later in this chapter.

1.0 DATA LINK CONFIGURATIONS

The serial interface is capable of operating in three serial data link configurations:

1) Half-Duplex, point-to-point
2) Half-Duplex, multipoint (with a half-duplex or full-duplex primary)
3) Loop

Figure 2 shows these three configurations. The RTS (Request to Send) and CTS (Clear to Send) hand-shaking signals are available in the point-to-point and multipoint configurations.

2.0 DATA CLOCKING OPTIONS

The serial interface can operate in an externally clocked mode or in a self clocked mode.

Externally Clocked Mode

In the externally clocked mode, a common Serial Data Clock (SCLK on pin 15) synchronizes the serial bit stream. This clock signal may come from the master CPU or primary station, or from an external phase-locked loop local to the 8044. Figure 3 illustrates the timing relationships for the serial interface signals when the externally clocked mode is used in point-to-point and multipoint data link configurations.

Incoming data is sampled at the rising edge of SCLK, and outgoing data is shifted out at the falling edge of SCLK. More detailed timing information is given in the 8044 data sheet.

Self Clocked (Asynchronous) Mode

The self clocked mode allows data transfer without a common system data clock. Using an on-chip DPLL (digital phase locked loop) the serial interface recovers the data clock from the data stream itself. The DPLL requires a reference clock equal to either 16 times or 32 times the data rate. This reference clock may be externally supplied or internally generated. When the serial interface generates this clock internally, it uses either the 8044's internal logic clock (half the crystal frequency's PH2) or the "timer 1" overflow. Figure 4 shows the serial interface signal timing relationships for the loop configuration, when the unclocked mode is used.

The DPLL monitors the received data in order to derive a data clock that is centered on the received bits. Centering is achieved by detecting all transitions of the received data, and then adjusting the clock transition (in increments of $1/16$ bit period) toward the center of the received bit. The DPLL converges to the nominal bit center within eight bit transitions, worst case.

To aid in the phase locked loop capture process, the 8044 has a NRZI (non-return-to-zero inverted) data encoding and decoding option. NRZI coding specifies that a signal does not change state for a transmitted binary 1, but does change state for a binary 0. Using the NRZI coding with zero-bit insertion, it can be guaranteed that an active signal line undergoes a transition at least every six bit times.

3.0 DATA RATES

The maximum data rate in the externally clocked mode is 2.4M bits per second (bps) a half-duplex configuration, and 1.0M in a loop configuration.

Order Number: 296165-001

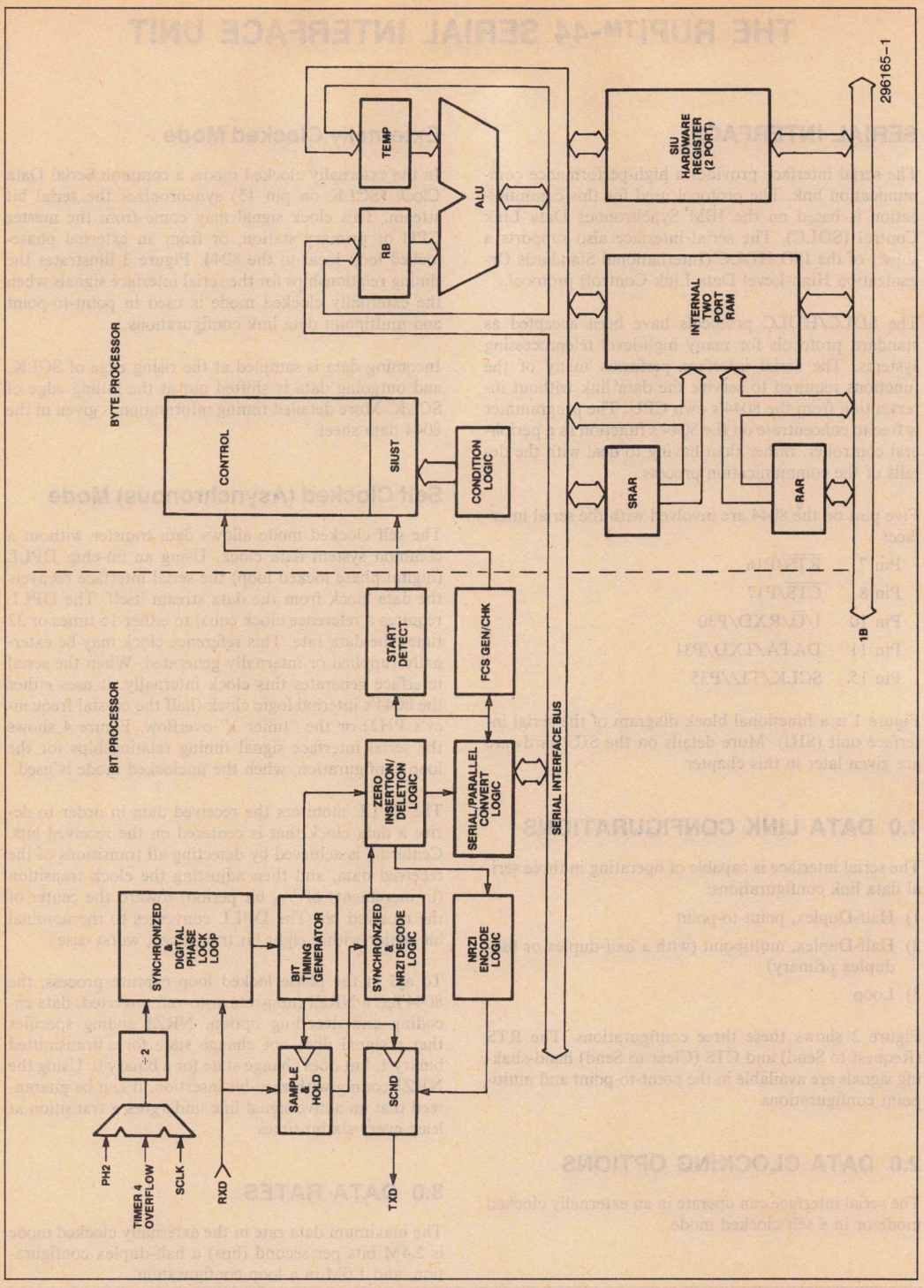
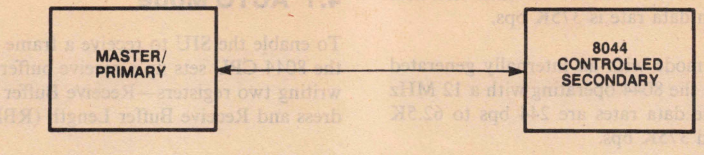
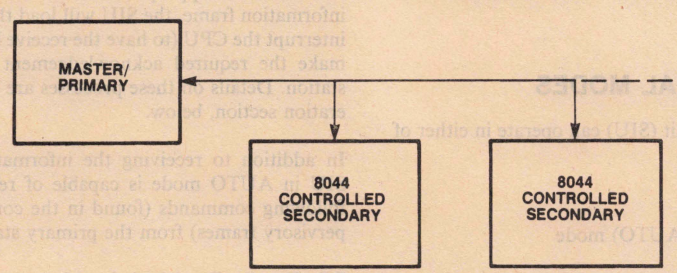


Figure 1. SIU Block Diagram



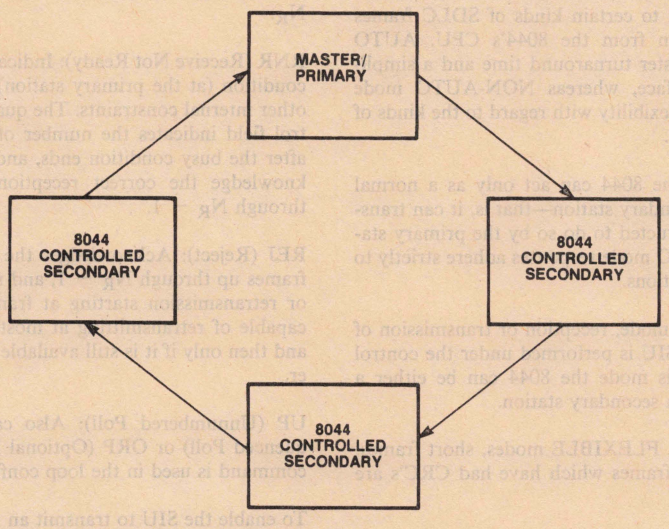
296165-2

1) HALF-DUPLEX, POINT-TO-POINT



296165-3

2) HALF-DUPLEX, MULTIPOINT



296165-4

3) LOOP

Figure 2. RUPI-44 Data Link Configurations

In the self clocked mode with an external reference clock, the maximum data rate is 375K bps.

In the self clocked mode with an internally generated reference clock, and the 8044 operating with a 12 MHz crystal, the available data rates are 244 bps to 62.5K bps, 187.5K bps and 375K bps.

For more details see the table in the SMD register description, below.

4.0 OPERATIONAL MODES

The Serial Interface Unit (SIU) can operate in either of two response modes:

1) AUTO mode
2) FLEXIBLE (NON-AUTO) mode

In the AUTO mode, the SIU performs in hardware a subset of the SDLC protocol called the normal response mode. The AUTO mode enables the SIU to recognize and respond to certain kinds of SDLC frames without intervention from the 8044's CPU. AUTO mode provides a faster turnaround time and a simplified software interface, whereas NON-AUTO mode provides a greater flexibility with regard to the kinds of operation permitted.

In AUTO mode, the 8044 can act only as a normal response mode secondary station—that is, it can transmit only when instructed to do so by the primary station. All such AUTO mode responses adhere strictly to IBM's SDLC definitions.

In the FLEXIBLE mode, reception or transmission of each frame by the SIU is performed under the control of the CPU. In this mode the 8044 can be either a primary station or a secondary station.

In both AUTO and FLEXIBLE modes, short frames, aborted frames, or frames which have had CRC's are ignored by the SIU.

The basic format of an SDLC frame is as follows:

Flag	Address	Control	Information	FCS	Flag

Format variations consist of omitting one or more of the fields in the SDLC frame. For example, a supervisory frame is formed by omitting the information field. Supervisory frames are used to confirm received frames, indicate ready or busy conditions, and to report errors. More details on frame formats are given in the SDLC Frame Format Options section, below.

4.1 AUTO Mode

To enable the SIU to receive a frame in AUTO mode, the 8044 CPU sets up a receive buffer. This is done by writing two registers—Receive Buffer Start (RBS) Address and Receive Buffer Length (RBL).

The SIU receives the frame, examines the control byte, and takes the appropriate action. If the frame is an information frame, the SIU will load the receive buffer, interrupt the CPU (to have the receive buffer read), and make the required acknowledgement to the primary station. Details on these processes are given in the Operation section, below.

In addition to receiving the information frames, the SIU in AUTO mode is capable of responding to the following commands (found in the control field of supervisory frames) from the primary station:

RR (Receive Ready): Acknowledges that the Primary station has correctly received numbered frames up through $N_R - 1$, and that it is ready to receive frame N_R.

RNR (Receive Not Ready): Indicates a temporary busy condition (at the primary station) due to buffering or other internal constraints. The quantity N_R in the control field indicates the number of the frame expected after the busy condition ends, and may be used to acknowledge the correct reception of the frames up through $N_R - 1$.

REJ (Reject): Acknowledges the correct reception of frames up through $N_R - 1$, and requests transmission or retransmission starting at frame N_R. The 8044 is capable of retransmitting at most the previous frame, and then only if it is still available in the transmit buffer.

UP (Unnumbered Poll): Also called NSP (Non-Sequenced Poll) or ORP (Optional Response Poll). This command is used in the loop configuration.

To enable the SIU to transmit an information frame in AUTO mode, the CPU sets up a transmit buffer. This is done by writing two registers—Transmit Buffer Start (TBS) Address and Transmit Buffer Length (TBL), and filling the transmit buffer with the information to be transmitted.

When the transmit buffer is full, the SIU can automatically (without CPU intervention) send an information frame (I-frame) with the appropriate sequence numbers, when the data link becomes available (when the 8044 is polled for information). After the SIU has transmitted the I-frame, it waits for acknowledgement from the receiving station. If the acknowledgement is

negative, the SIU retransmits the frame. If the acknowledgement is positive, the SIU interrupts the CPU, to indicate that the transmit buffer may be reloaded with new information.

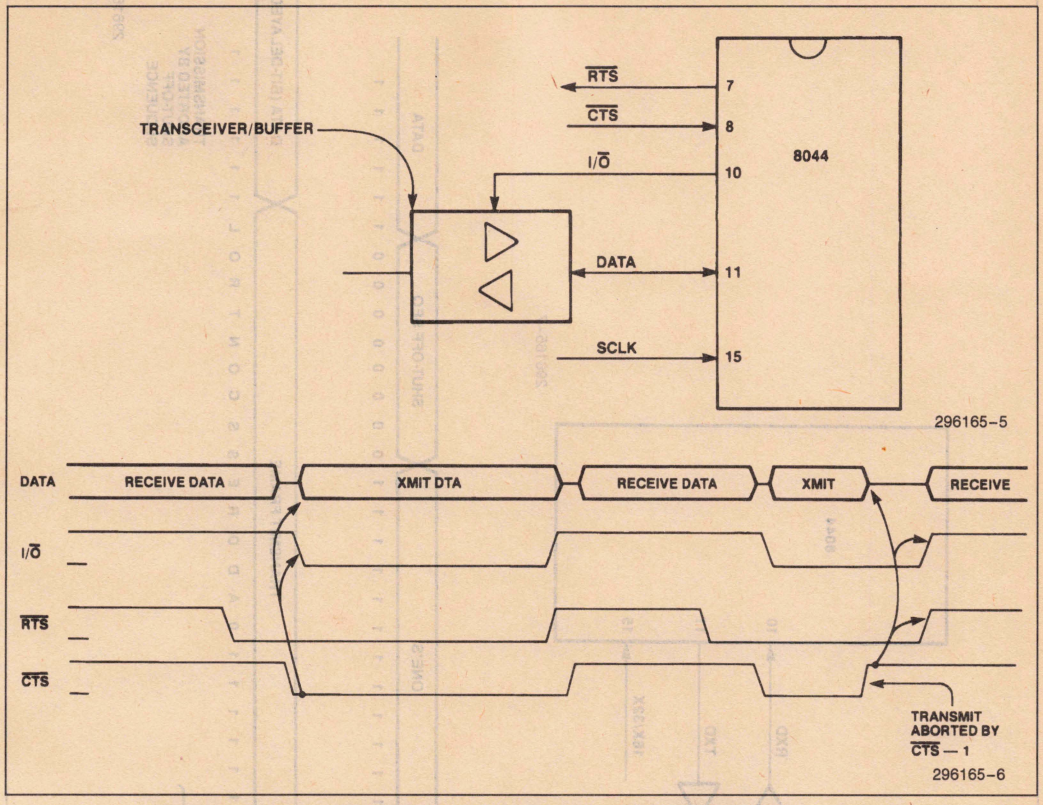


Figure 3. Serial Interface Timing—Clocked Mode

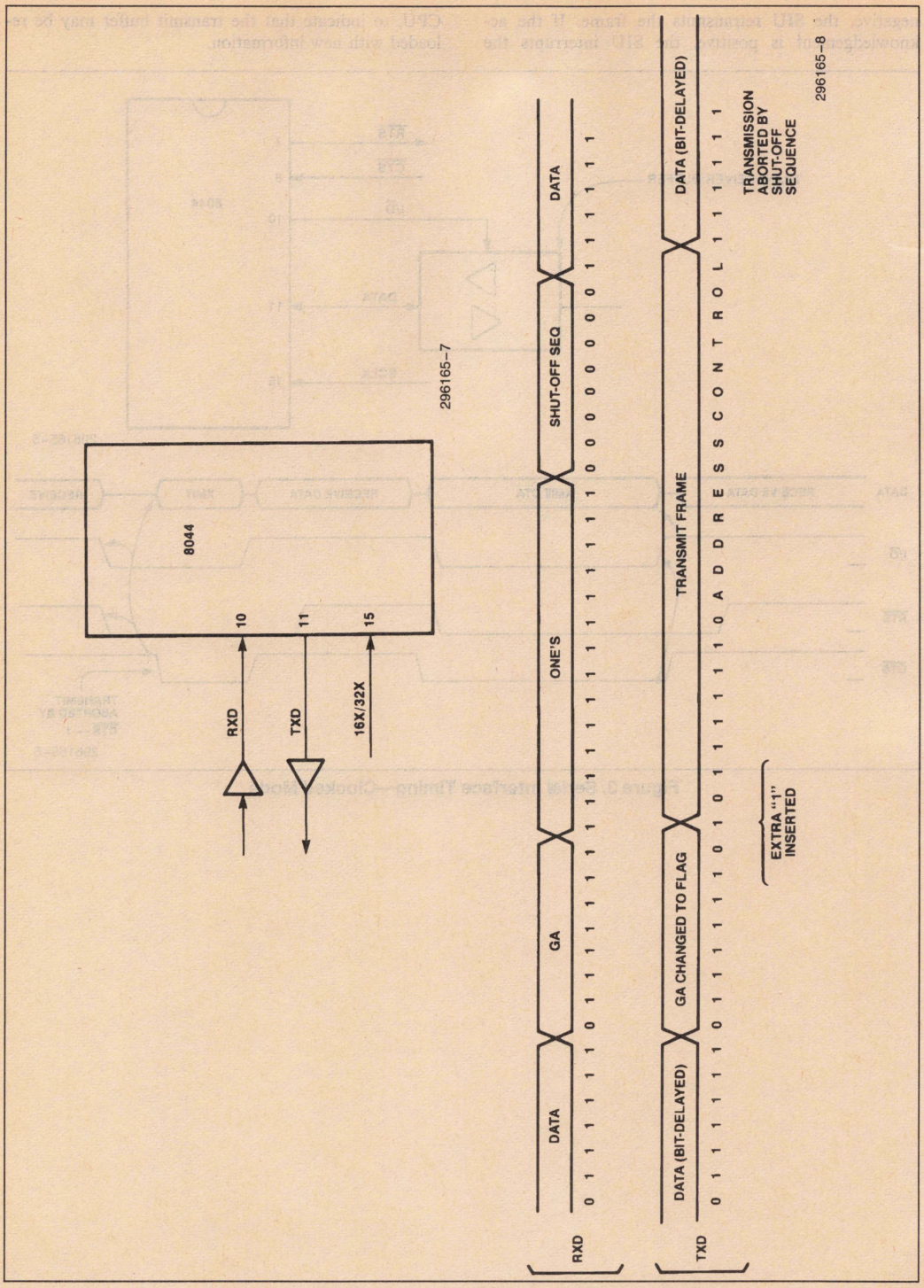


Figure 4. Serial Interface Timing—Self Clocked Mode

In addition to transmitting the information frames, the SIU in AUTO mode is capable of sending the following responses to the primary station:

RR (Receive Ready): Acknowledges that the 8044 has correctly received numbered frames up through $N_R - 1$, and that it is ready to receive frame N_R.

RNR (Receive Not Ready): Indicates a temporary busy condition (at the 8044) due to buffering or other internal constraints. The quantity N_R in the control field indicates the number of the frame expected after the busy condition ends, and acknowledges the correct reception of the frames up through $N_R - 1$.

4.2 FLEXIBLE Mode

In the FLEXIBLE (or non-auto) mode, all reception and transmission is under the control of the CPU. The full SDLC and HDLC protocols can be implemented, as well as any bit-synchronous variants of these protocols.

FLEXIBLE mode provides more flexibility than AUTO mode, but it requires more CPU overhead, and much longer recognition and response times. This is especially true when the CPU is servicing an interrupt that has higher priority than the interrupts from the SIU.

In FLEXIBLE mode, when the SIU receives a frame, it interrupts the CPU. The CPU then reads the control byte from the Receive Control Byte (RCB) register. If the received frame is an information frame, the CPU also reads the information from the receive buffer, according to the values in the Receive Buffer Start (RBS) address register and the Received Field Length (RFL) register.

In FLEXIBLE mode, the 8044 can initiate transmissions without being polled, and thus it can act as the primary station. To initiate transmission or to generate a response, the CPU sets up and enables the SIU. The SIU then formats and transmits the desired frame. Upon completion of the transmission, without waiting for a positive acknowledgement from the receiving station, the SIU interrupts the CPU.

5.0 8044 FRAME FORMAT OPTIONS

As mentioned above, variations on the basic SDLC frame consist of omitting one or more of the fields. The choice of which fields to omit, as well as the selection of AUTO mode versus FLEXIBLE mode, is specified by the settings of the following three bits in the Serial Mode Register (SMD) and the Status/Control Register (STS):

SMD Bit 0: NFCS (No Frame Check Sequence)

SMD Bit 1: NB (Non-Buffered Mode—No Control Field)

STS Bit 1: AM (AUTO Mode or Addressed Mode)

Figure 5 shows how these three bits control the frame format.

The following paragraphs discuss some properties of the standard SDLC format, and the significance of omitting some of the fields.

5.1 Standard SDLC Format

The standard SDLC format consists of an opening flag, an 8-bit address field, an 8-bit control field, an n-byte information field, a 16-bit Frame Check Sequence (FCS), and a closing flag. The FCS is based on the CCITT-CRC polynominal ($X^{16} + X^{12} + X^5 + 1$). The address and control fields may not be extended. Within the 8044, the address field is held in the Station Address (STAD) register, and the control field is held in the Receive Control Byte (RCB) or Transmit Control Byte (TCB) register. The standard SDLC format may be used in either AUTO mode or FLEXIBLE mode.

5.2 No Control Field (Non-Buffered Mode)

When the control field is not present, the RCB and TCB registers are not used. The information field begins immediately after the address field, or, if the address field is also absent, immediately after the opening flag. The entire information field is stored in the 8044's on-chip RAM. If there is no control field, FLEXIBLE mode must be used. Control information may, of course, be present in the information field, and in this manner the No Control Field option may be used for implementing extended control fields.

5.3 No Control Field and No Address Field

The No Address Field option is available only in conjuction with the No Control Field option. The STAD, RCB, and TCB registers are not used. When both these fields are absent, the information field begins immediately after the opening flag. The entire information field is stored in on-chip RAM. FLEXIBLE mode must be used. Formats without an address field have the following applications:

Point-to-point data links (where no addressing is necessary)

Monitoring line activity (receiving all messages regardless of the address field)

Extended addressing

FRAME OPTION	NFCS	NB	AM	FRAME FORMAT
Standard SDLC FLEXIBLE Mode	0	0	0	F A C I FCS F
Standard SDLC AUTO Mode	0	0	1	F A C I FCS F
No Control Field FLEXIBLE Mode	0	1	1	F A I FCS F
No Control Field No Address Field FLEXIBLE Mode	0	1	0	F I FCS F
No FCS Field FLEXIBLE Mode	1	0	0	F A C I F
No FCS Field AUTO Mode	1	0	1	F A C I F
No FCS Field No Control Field FLEXIBLE Mode	1	1	1	F A I F
No FCS Field No Control Field No Address Field FLEXIBLE Mode	1	1	0	F I F

Key to Abbreviations:

F = Flag (01111110)
A = Address Field
C = Control Field
I = Information Field
FCS = Frame Check Sequence

NOTE:
The AM bit is AUTO mode control bit when NB = 0, and Address Mode control bit when NB = 1.

Figure 5. Frame Format Options

5.4 No FCS Field

In the normal case (NFCS = 0), the last 16 bits before the closing flag are the Frame Check Sequence (FCS) field. These bits are not stored in the 8044's RAM. Rather, they are used to compute a cyclic redundancy check (CRC) on the data in the rest of the frame. A received frame with a CRC error (incorrect FCS) is ignored. In transmission, the FCS field is automatically computed by the SIU, and placed in the transmitted frame just prior to the closing flag.

The NFCS bit (SMD Bit 0) gives the user the capability of overriding this automatic feature. When this bit is set (NFCS = 1), all bits from the beginning of the information field to the beginning of the closing flag are treated as part of the information field, and are stored in the on-chip RAM. No FCS checking is done on the received frames, and no FCS is generated for the transmitted frames. The No FCS Field option may be used in conjunction with any of the other options. It is typically used in FLEXIBLE mode, although it does not strictly include AUTO mode. Use of the No FCS Field option AUTO Mode may, however, result in SDLC protocol violations, since the data integrity is not checked by the SIU.

Formats without an FCS field have the following applications:

Receiving and transmitting frames without verifying data integrity.

Using an alternate data verification algorithm.

Using an alternate CRC-16 polynomial (such as $X^{16} + X^{15} + X^2 + 1$), or a 32-bit CRC

Performing data link diagnosis by forcing false CRCs to test error detection mechanisms

In addition to the applications mentioned above, all of the format variations are useful in the support of non-standard bit-synchronous protocols.

6.0 HDLC

In addition to its support of SDLC communications, the 8044 also supports some of the capabilities of HDLC. The following remarks indicate the principal differences between SDLC and HDLC.

HDLC permits any number of bits in the information field, whereas SDLC requires a byte structure (multiple of 8 bits). The 8044 itself operates on byte boundaries, and thus it restricts fields to multiples of 8 bits.

HDLC provides functional extensions to SDLC: an unlimited address field is allowed, and extended frame number sequencing.

HDLC does not support operation in loop configurations.

7.0 SIU SPECIAL FUNCTION REGISTERS

The 8044 CPU communicates with and controls the SIU through hardware registers. These registers are accessed using direct addressing. The SIU special function registers (SIU SFRs) are of three types:

Control and Status Registers

Parameter Registers

ICE Support Registers

7.1 Control and Status Registers

There are three SIU Control and Status Registers:

Serial Mode Register (SMD)

Status/Command Register (STS)

Send/Receive Count Register (NSNR)

The SMD, STS, and NSNR registers are all cleared by system reset. This assures that the SIU will power up in an idle state (neither receiving nor transmitting).

These registers and their bit assignments are described below (see also the More Details on Registers section).

SMD: SERIAL MODE REGISTER (BYTE-ADDRESSABLE)

Bit: 7 6 5 4 3 2 1 0

SCM2	SCM1	SCM0	NRZI	LOOP	PFS	NB	NFCS

The Serial Mode Register (Address C9H) selects the operational modes of the SIU. The 8044 CPU can both read and write SMD. The SIU can read SMD but cannot write to it. To prevent conflict between CPU and SIU access to SMD, the CPU should write SMD only when the Request To Send (RTS) and Receive Buffer Empty (RBE) bits (in the STS register) are both false (0). Normally, SMD is accessed only during initialization.

The individual bits of the Serial Mode Register are as follows:

Bit #	Name	Description
SMD.0	NFCS	No FCS field in the SDLC frame.
SMD.1	NB	Noon-Buffered mode. No control field in the SDLC frame.
SMD.2	PFS	Pre-Frame Sync mode. In this mode, the 8044 transmits two bytes before the first flag of a frame, for DPLL synchronization. If NRZI is enabled, 00H is sent; otherwise, 55H is sent. In either case, 16 pre-frame transitions are guaranteed.
SMD.3	LOOP	Loop configuration.
SMD.4	NRZI	NRZI coding option.
SMD.5	SCM0	Select Clock Mode—Bit 0
SMD.6	SCM1	Select Clock Mode—Bit 1
SMD.7	SCM2	Select Clock Mode—Bit 2

The SCM bits decode as follows:

SCM 2 1 0	Clock Mode	Data Rate (Bits/sec)*
0 0 0	Externally clocked	0–2.4M**
0 0 1	Undefined	
0 1 0	Self clocked, timer overflow	244–62.5K
0 1 1	Undefined	
1 0 0	Self clocked, external 16x	0–375K
1 0 1	Self clocked, external 32x	0–187.5K
1 1 0	Self clocked, internal fixed	375K
1 1 1	Self clocked, internal fixed	187.5K

*Based on a 12 MHz crystal frequency
**0–1M bps in loop configuration

STS: STATUS/COMMAND REGISTER (BIT-ADDRESSABLE)

Bit:	7	6	5	4	3	2	1	0
	TBF	RBE	RTS	SI	BOV	OPB	AM	RBP

The Status/Command Register (Address C8H) provides operational control of the SIU by the 8044 CPU, and enables the SIU to post status information for the CPU's access. The SIU can read STS, and can alter certain bits, as indicated below. The CPU can both read and write STS asynchronously. However, 2-cycle instructions that access STS during both cycles ('JBC/B, REL' and 'MOV /B,C.') should not be used, since the SIU may write to STS between the two CPU accesses.

The individual bits of the Status/Command Register are as follows:

Bit#	Name	Description
STS.0	RBP	Receive Buffer Protect. Inhibits writing of data into the receive buffer. In AUTO mode, RBP forces an RNR response instead of an RR.
STS.1	AM	AUTO Mode/Addressed Mode. Selects AUTO mode where AUTO mode is allowed. If NB is true, (= 1), the AM bit selects the addressed mode. AM may be cleared by the SIU.
STS.2	OPB	Optional Poll Bit. Determines whether the SIU will generate an AUTO response to an optional poll (UP with P = 0). OPB may be set or cleared by the SIU.
STS.3	BOV	Receive Buffer Overrun. BOV may be set or cleared by the SIU.
STS.4	SI	SIU Interrupt. This is one of the five interrupt sources to the CPU. The vector location = 23H. SI may be set by the SIU. It should be cleared by the CPU before returning from an interrupt routine.
STS.5	RTS	Request To Send. Indicates that the 8044 is ready to transmit or is transmitting. RTS may be read or written by the CPU. RTS may be read by the SIU, and in AUTO mode may be written by the SIU.
STS.6	RBE	Receive Buffer Empty. RBE can be thought of as Receive Enable. RBE is set to one by the CPU when it is ready to receive a frame, or has just read the buffer, and to zero by the SIU when a frame has been received.
STS.7	TBF	Transmit Buffer Full. Written by the CPU to indicate that it has filled the transmit buffer. TBF may be cleared by the SIU.

NSNR: SEND/RECEIVE COUNT REGISTER (BIT-ADDRESSABLE)

Bit:	7	6	5	4	3	2	1	0
	NS2	NS1	NS0	SES	NR2	NR1	NR0	SER

The Send/Receive Count Register (Address D8H) contains the transmit and receive sequence numbers, plus tally error indications. The SIU can both read and write NSNR. The 8044 CPU can both read and write NSNR asynchronously. However, 2-cycle instructions that access NSNR during both cycles ('JBC /B, REL', and 'MOV /B,C') should not be used, since the SIU may write to NSNR between the two 8044 CPU accesses.

The individual bits of the Send/Receive Count Register are as follows:

Bit#	Name	Description
NSNR.0	SER	Receive Sequence Error: NS (P) ≠ NR (S)
NSNR.1	NR0	Receive Sequence Counter—Bit 0
NSNR.2	NR1	Receive Sequence Counter—Bit 1
NSNR.3	NR2	Receive Sequence Counter—Bit 2
NSNR.4	SES	Send Sequence Error: NR (P) ≠ NS (S) and NR (P) ≠ NS (S) + 1
NSNR.5	NS0	Send Sequence Counter—Bit 0
NSNR.6	NS1	Send Sequence Counter—Bit 1
NSNR.7	NS2	Send Sequence Counter—Bit 2

7.2 Parameter Registers

There are eight parameter registers that are used in connection with SIU operation. All eight registers may be read or written by the 8044 CPU. RFL and RCB are normally loaded by the SIU.

The eight parameter registers are as follows:

STAD: STATION ADDRESS REGISTER (BYTE-ADDRESSABLE)

The Station Address register (Address CEH) contains the station address. To prevent access conflict, the CPU

should access STAD only when the SIU is idle (RTS = 0 and RBE = 0). Normally, STAD is accessed only during initialization.

TBS: TRANSMIT BUFFER START ADDRESS REGISTER (BYTE-ADDRESSABLE)

The Transmit Buffer Start address register (Address DCH) points to the location in on-chip RAM for the beginning of the I-field of the frame to be transmitted. The CPU should access TBS only when the SIU is not transmitting a frame (when TBF = 0).

TBL: TRANSMIT BUFFER LENGTH REGISTER (BYTE-ADDRESSABLE)

The Transmit Buffer Length register (Address DBH) contains the length (in bytes) of the I-field to be transmitted. A blank I-field (TBL = 0) is valid. The CPU should access TBL only when the SIU is not transmitting a frame (when TBF = 0).

NOTE:
The transmit and receive buffers are not allowed to "wrap around" in the on-chip RAM. A "buffer end" is automatically generated if address 191 (BFH) is reached.

TCB: TRANSMIT CONTROL BYTE REGISTER (BYTE-ADDRESSABLE)

The Transmit Control Byte register (Address DAH) contains the byte which is to be placed in the control field of the transmitted frame, during NON-AUTO mode transmission. The CPU should access TCB only when the SIU is not transmitting a frame (when TBF = 0). The N_S and N_R counters are not used in the NON-AUTO mode.

RBS: RECEIVE BUFFER START ADDRESS REGISTER (BYTE-ADDRESSABLE)

The Receive Buffer Start address register (Address CCH) points to the location in on-chip RAM where the beginning of the I-field of the frame being received is to be stored. The CPU should write RBS only when the SIU is not receiving a frame (when RBE = 0).

RBL: RECEIVE BUFFER LENGTH REGISTER (BYTE-ADDRESSABLE)

The Receive Buffer Length register (Address CBH) contains the length (in bytes) of the area in on-chip RAM allocated for the received I-field. RBL = 0 is valid. The CPU should write RBL only when RBE = 0.

RFL: RECEIVE FIELD LENGTH REGISTER (BYTE-ADDRESSABLE)

The Received Field Length register (Address CDH) contains the length (in bytes) of the received I-field that has just been loaded into on-chip RAM. RFL is loaded by the SIU. RFL = 0 is valid. RFL should be accessed by the CPU only when RBE = 0.

RCB: RECEIVE CONTROL BYTE REGISTER (BYTE-ADDRESSABLE)

The Received Control Byte register (Address CAH) contains the control field of the frame that has just been received. RCB is loaded by the SIU. The CPU can only read RCB, and should only access RCB when RBE = 0.

7.3 ICE Support Registers

The 8044 In-Circuit Emulator (ICE-44) allows the user to exercise the 8044 application system and monitor the execution of instructions in real time.

The emulator operates with Intel's Intellec® development system. The development system interfaces with the user's 8044 system through an in-cable buffer box. The cable terminates in a 8044 pin-compatible plug, which fits into the 8044 socket in the user's system. With the emulator plug in place, the user can exercise his system in real time while collecting up to 255 instruction cycles of real-time data. In addition, he can single-step the program.

Static RAM is available (in the in-cable buffer box) to emulate the 8044 internal and external program memory and external data memory. The designer can display and alter the contents of the replacement memory in the buffer box, the internal data memory, and the internal 8044 registers, including the SFRs.

Among the SIU SFRs are the following registers that support the operation of the ICE:

DMA CNT: DMA COUNT REGISTER (BYTE-ADDRESSABLE)

The DMA Count register (Address CFH) indicates the number of bytes remaining in the information block that is currently being used.

FIFO: THREE-BYTE (BYTE-ADDRESSABLE)

The Three-Byte FIFO (Address DDH, DEH, and DFH) is used between the eight-bit shift register and the information buffer when an information block is received.

SIUST: SIU STATE COUNTER (BYTE-ADDRESSABLE)

The SIU State Counter (Address D9H) reflects the state of the internal logic which is under SIU control. Therefore, care must be taken not to write into this register.

The SIUST register can serve as a helpful aid to determine which field of a receive frame that the SIU expects next. The table below will help in debugging 8044 reception problems.

SIUST Value	Function
01H	Waiting for opening flag.
08H	Waiting for address field.
10H	Waiting for control field.
18H	Waiting for first byte of I field. This state is only entered if a FCS is expected. It pushes the received byte onto the top of the FIFO.
20H	Waiting for second byte of I field. This state always follows state 18H.
28H	Waiting for I field byte. This state can be entered from state 20H or from states 01H, 08H, or 10H depending upon the SIU's mode configuration. (Each time a byte is received, it is pushed onto the top of the FIFO and the byte at the bottom is put into memory. For no FCS formatted frames, the FIFO is collapsed into a single register).
30H	Waiting for the closing flag after having overflowed the receive buffer. Note that even if the receive frame overflows the assigned receive buffer length, the FCS is still checked.

Examples of SIUST status sequences for different frame formats are shown below. Note that status changes after acceptance of the received field byte.

Table 1. SIUST Status Sequences

										Frame Option		
										NFCS	NB	AM
Example 1:												
Frame Format	(Idle)	F	A	C	I			FCS	F	0	0	1
SIUST Value	01	01	08	10	18	20	28	28	01			
Example 2:												
Frame Format	(Idle)	F	A	I			FCS	F		0	1	1
SIUST Value	01	01	08	18	20	28	28	01				
Example 3:												
Frame Format	(Idle)	F	I			FCS	F			0	1	0
SIUST Value	01	01	18	20	28	28	01					
Example 4:												
Frame Format	(Idle)	F	A	I	F					1	1	1
SIUST Value	01	01	08	28	01							
Example 5:												
Frame Format	(Idle)	F	I		F					1	1	0
SIUST Value	01	01	28		01							
Example 6:												
Frame Format	(Idle)	F	I			I OVERFLOW		FCS	F	0	1	0
SIUST Value	01	01	18	20	28	30		30	01			

8.0 OPERATION

The SIU is initialized by a reset signal (on pin 9), followed by write operations to the SIU SFRs. Once initialized, the SIU can function in AUTO mode or NON-AUTO mode. Details are given below.

8.1 Initialization

Figure 6 is the SIU. Registers SMD, STS, and NSNR are cleared by reset. This puts the 8044 into an idle state—neither receiving nor transmitting. The following registers must be initialized before the 8044 leaves the idle state:

STAD — to establish the 8044's SDLC station address.

SMD — To configure the 8044 for the proper operating mode.

RBS, RBL — to define the area in RAM allocated for the Receive Buffer.

TBS, TBL — to define the area in RAM allocated for the Transmit Buffer.

Once these registers have been initialized, the user may write to the STS register to enable the SIU to leave the idle state, and to begin transmits and/or receives.

Setting RBE to 1 enables the SIU for receive. When RBE = 1, the SIU monitors the received data stream for a flag pattern. When a flag pattern is found, the SIU enters Receive mode and receives the frame.

Setting RTS to 1 enables the SIU for transmit. When RTS = 1, the SIU monitors the received data stream for a GA pattern (loop configuration) or waits for a CTS (non-loop configuration). When the GA or CTS arrives, the SIU enters Transmit mode and transmits a frame.

In AUTO mode, the SIU sets RTS to enable automatic transmissions of appropriate responses.

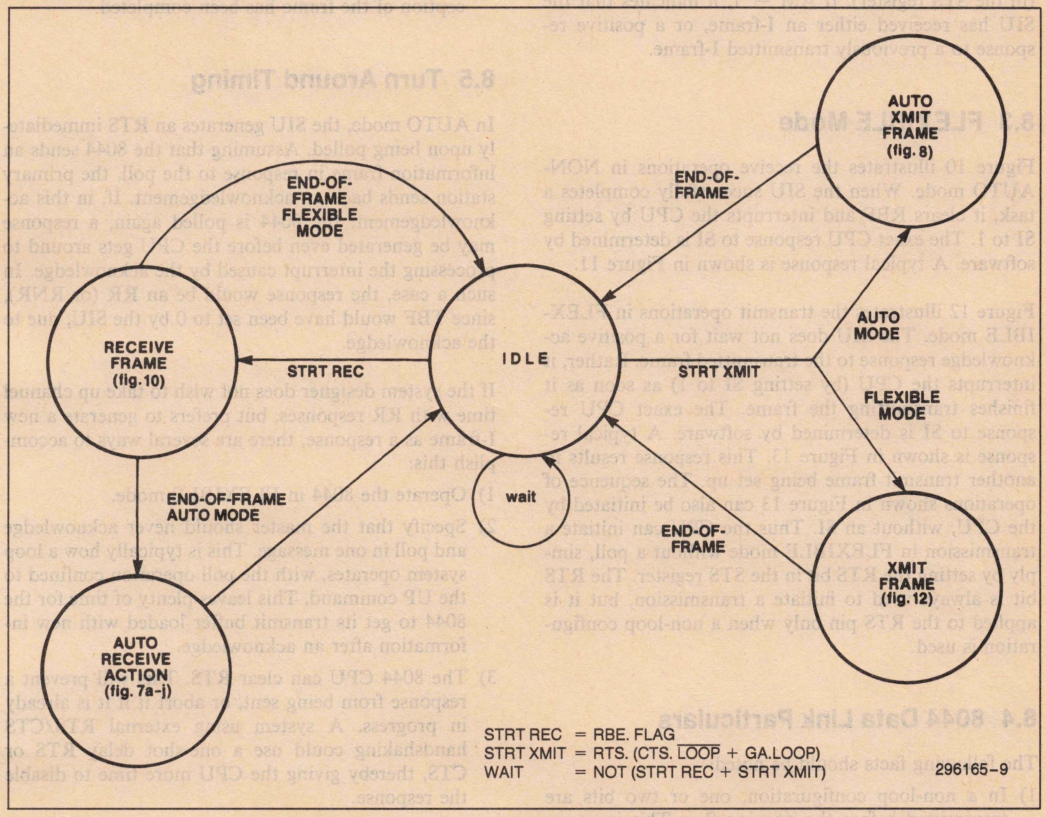


Figure 6. SIU State Diagram

8.2 AUTO Mode

Figure 7 illustrates the receive operations in AUTO mode. The overall operation is shown in Figure 7a. Particular cases are illustrated in Figures 7b through 7j. If any Unnumbered Command other than UP is received, the AM bit is cleared and the SIU responds as if in the FLEXIBLE mode, by interrupting the CPU for supervision. This will also happen if a BOV or SES condition occurs. If the received frame contains a poll, the SIU sets the RTS bit to generate a response.

Figure 8 illustrates the transmit operations in AUTO mode. When the SIU gets the opportunity to transmit, and if the transmit buffer is full, it sends an I-frame. Otherwise, it sends an RR if the buffer is free, or an RNR if the buffer is protected. The sequence counters NS and NR are used to construct the appropriate control fields.

Figure 9 shows how the CPU responds to an SI (serial interrupt) in AUTO mode. The CPU tests the AM bit (in the STS register). If AM = 1, it indicates that the SIU has received either an I-frame, or a positive response to a previously transmitted I-frame.

8.3 FLEXIBLE Mode

Figure 10 illustrates the receive operations in NON-AUTO mode. When the SIU successfully completes a task, it clears RBF and interrupts the CPU by setting SI to 1. The exact CPU response to SI is determined by software. A typical response is shown in Figure 11.

Figure 12 illustrates the transmit operations in FLEXIBLE mode. The SIU does not wait for a positive acknowledge response to the transmitted frame. Rather, it interrupts the CPU (by setting SI to 1) as soon as it finishes transmitting the frame. The exact CPU response to SI is determined by software. A typical response is shown in Figure 13. This response results in another transmit frame being set up. The sequence of operations shown in Figure 13 can also be initiated by the CPU, without an SI. Thus the CPU can initiate a transmission in FLEXIBLE mode without a poll, simply by setting the RTS bit in the STS register. The RTS bit is always used to initiate a transmission, but it is applied to the RTS pin only when a non-loop configuration is used.

8.4 8044 Data Link Particulars

The following facts should be noted:

1) In a non-loop configuration, one or two bits are transmitted before the opening flag. This is necessary for NRZI synchronization.
2) In a non-loop configuration, one to eight extra dribble bits are transmitted after the closing flag. These bits are a zero followed by ones.
3) In a loop configuration, when a GA is received and the 8044 begins transmitting, the sequence is 01111110101111110 ... (FLAG, 1, FLAG, ADDRESS, etc.). The first flag is created from the GA. The second flag begins the message.
4) CTS is sampled after the rising edge of the serial data, at about the center of the bit cell, except during a non-loop, externally clocked mode transmit, in which case it is sampled just after the falling edge.
5) The SIU does not check for illegal I-fields. In particular, if a supervisory command is received in AUTO mode, and if there is also an I-field, it will be loaded into the receive buffer (if RBP = 0), but it cannot cause a BOV.
6) In relation to the Receive Buffer Protect facility, the user should set RFL to 0 when clearing RBP, such that, if the SIU is in the process of receiving a frame, RFL will indicate the proper value when reception of the frame has been completed.

8.5 Turn Around Timing

In AUTO mode, the SIU generates an RTS immediately upon being polled. Assuming that the 8044 sends an information frame in response to the poll, the primary station sends back an acknowledgement. If, in this acknowledgement, the 8044 is polled again, a response may be generated even before the CPU gets around to processing the interrupt caused by the acknowledge. In such a case, the response would be an RR (or RNR), since TBF would have been set to 0 by the SIU, due to the acknowledge.

If the system designer does not wish to take up channel time with RR responses, but prefers to generate a new I-frame as a response, there are several ways to accomplish this:

1) Operate the 8044 in FLEXIBLE mode.
2) Specify that the master should never acknowledge and poll in one message. This is typically how a loop system operates, with the poll operation confined to the UP command. This leaves plenty of time for the 8044 to get its transmit buffer loaded with new information after an acknowledge.
3) The 8044 CPU can clear RTS. This will prevent a response from being sent, or abort it if it is already in progress. A system using external RTS/CTS handshaking could use a one-shot delay RTS or CTS, thereby giving the CPU more time to disable the response.

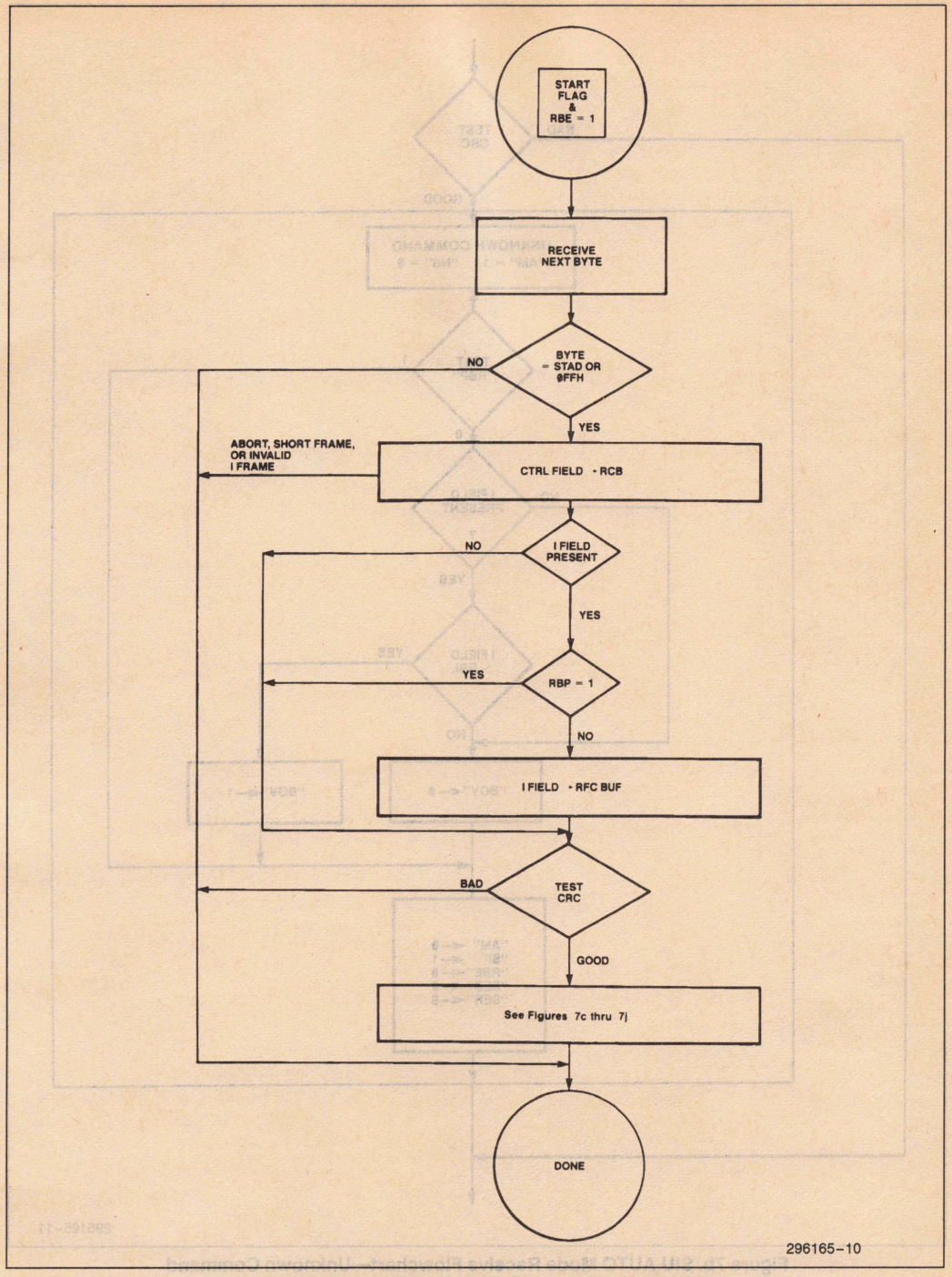


Figure 7a. SIU AUTO Mode Receive Flowchart—General

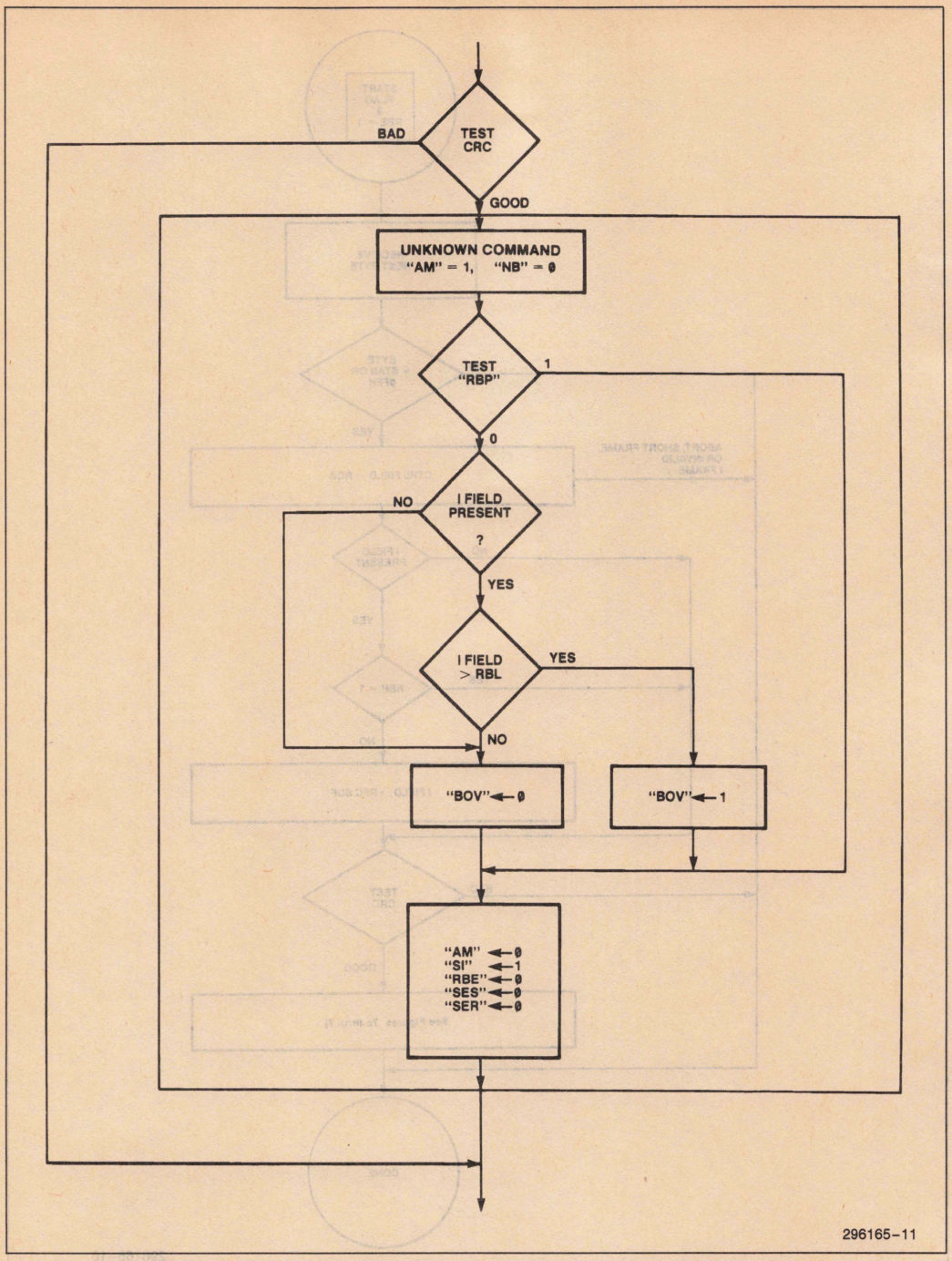


Figure 7b. SIU AUTO Mode Receive Flowchart—Unknown Command

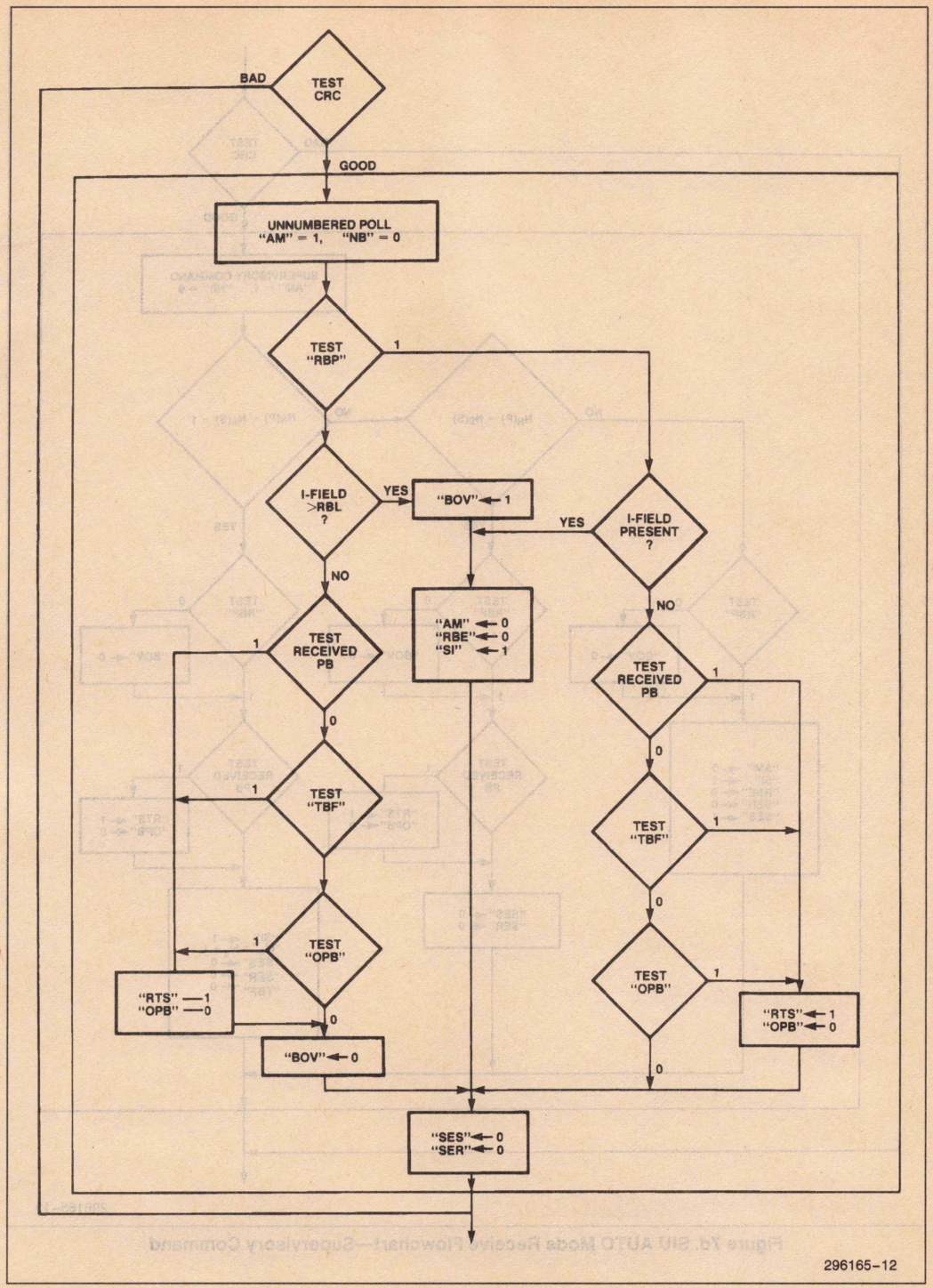


Figure 7c. SIU AUTO Mode Receive Flowchart—Unnumbered Poll

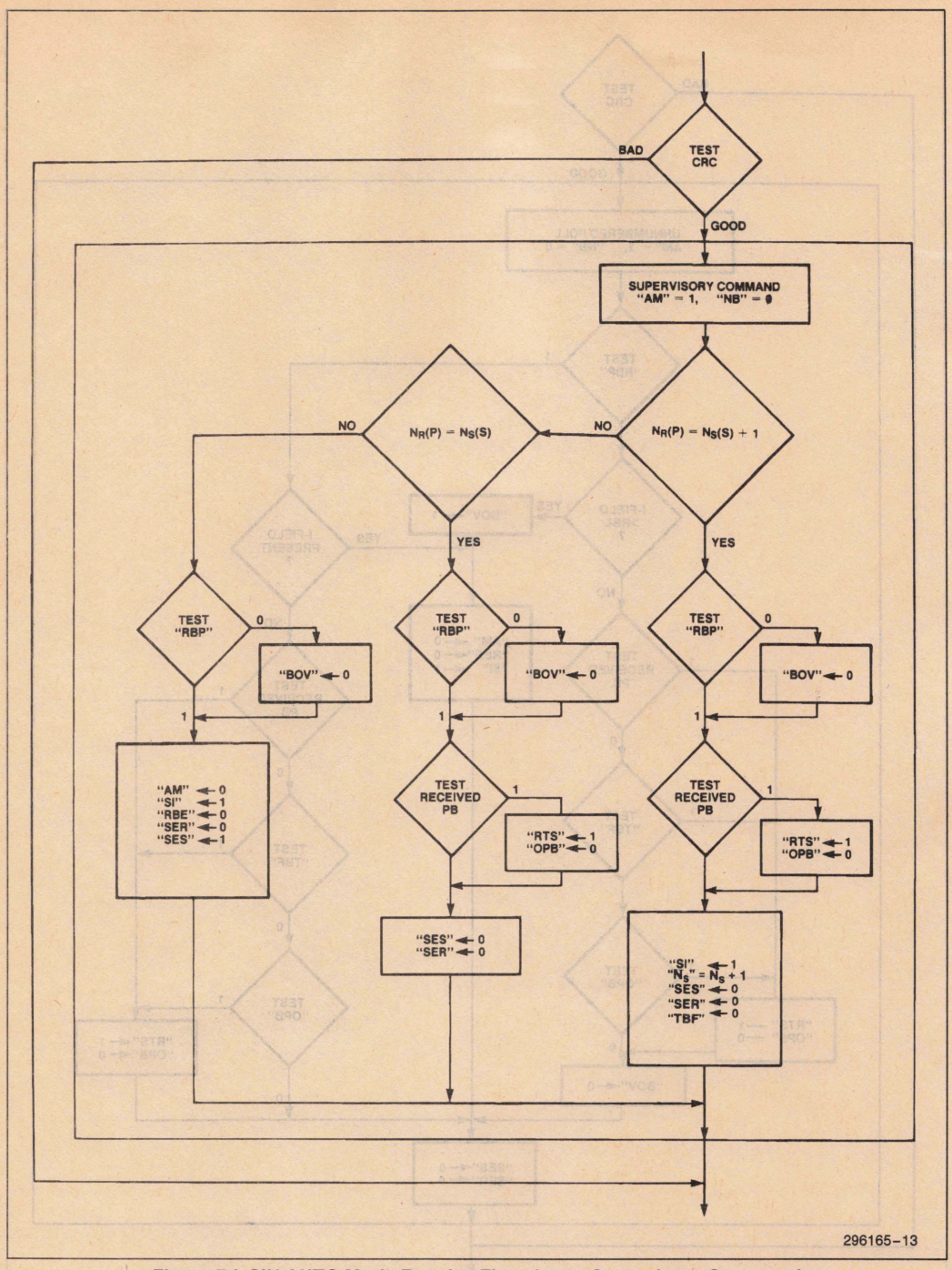


Figure 7d. SIU AUTO Mode Receive Flowchart—Supervisory Command

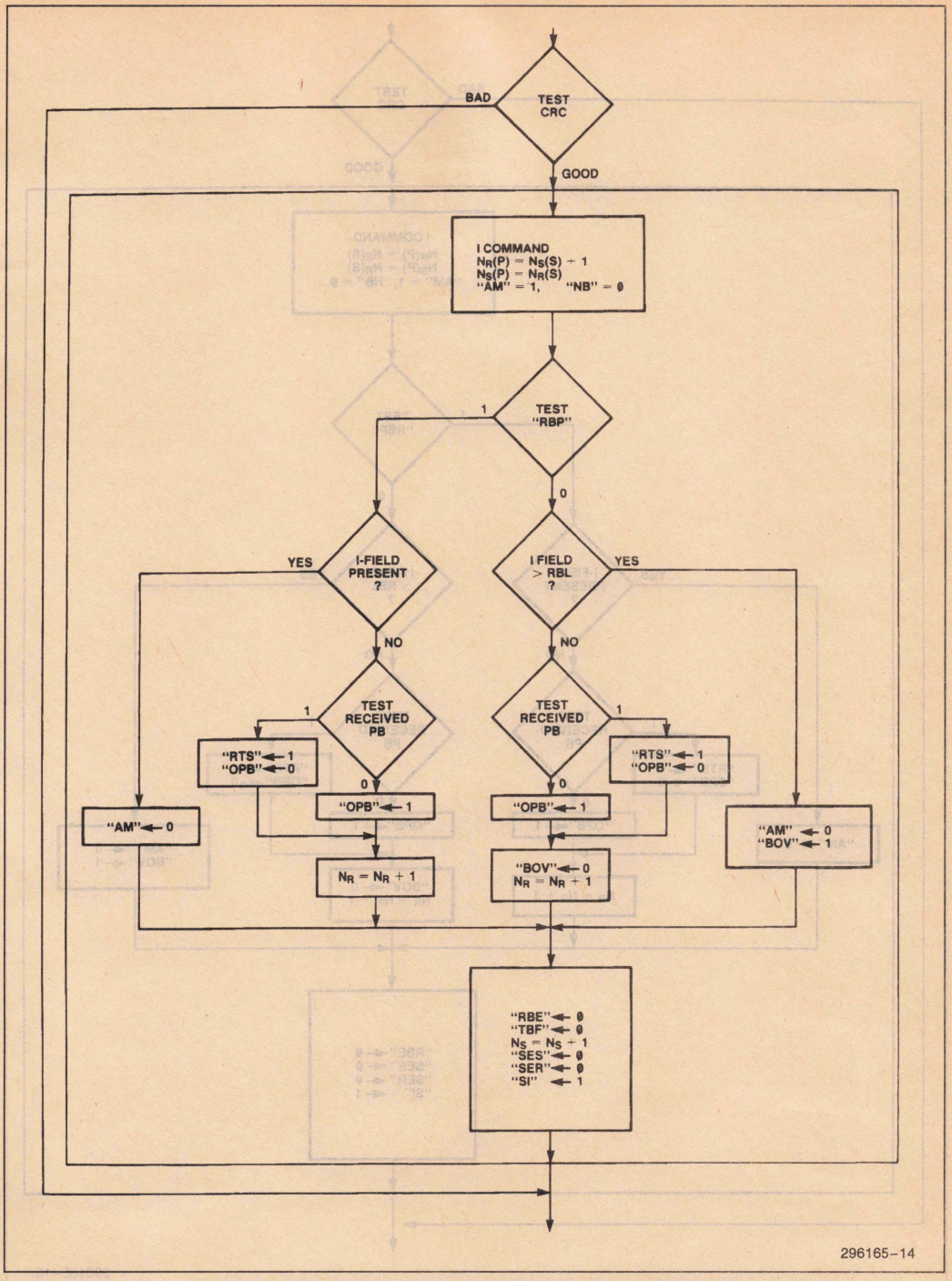


Figure 7e. SIU AUTO Mode Receive Flowchart—I Command: Prior Transmitted I-Field Confirmed, Current Received I-Field in Sequence

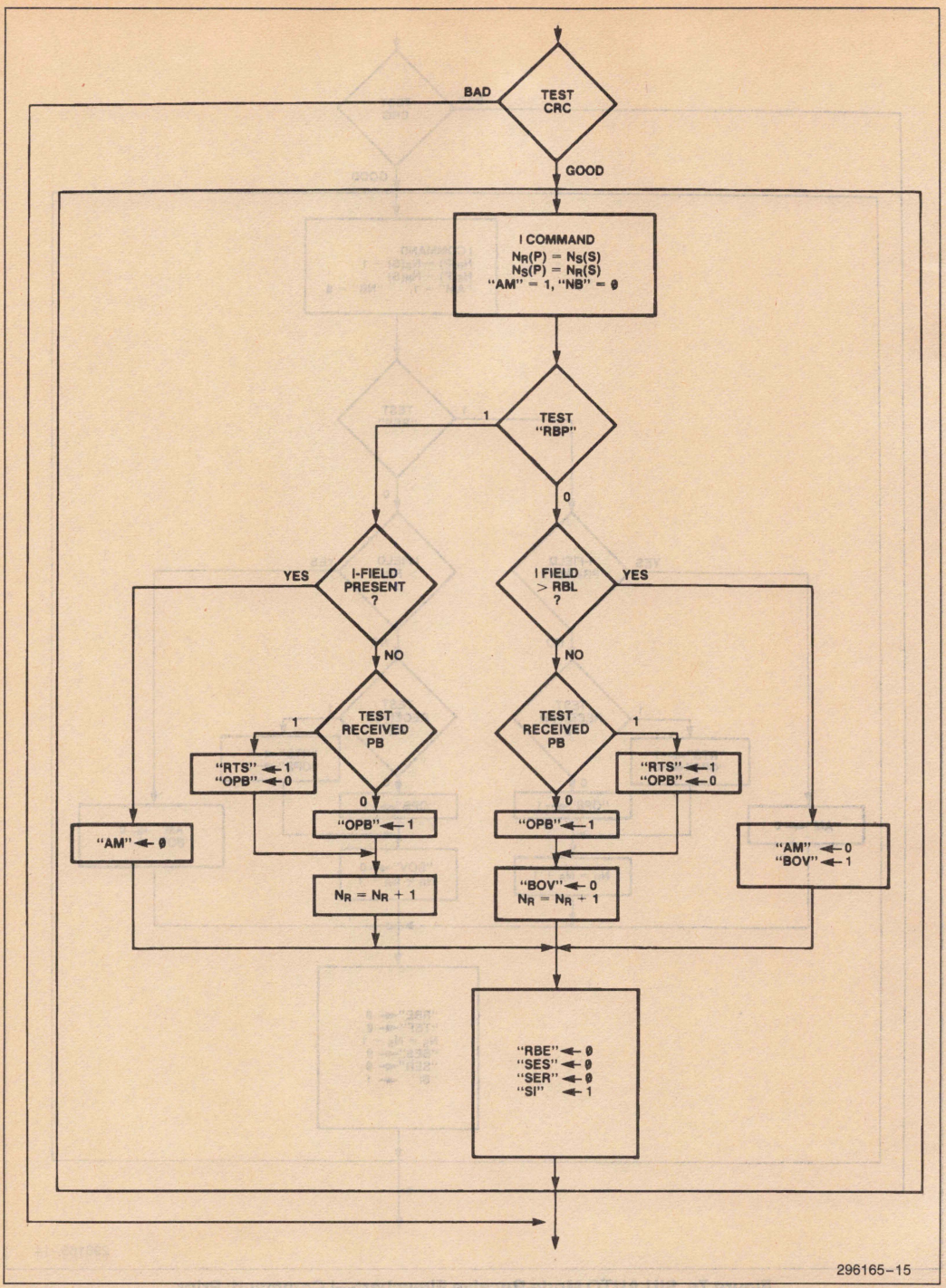


Figure 7f. SIU AUTO Mode Receive Flowchart—I Command: Prior Transmitted I-Field Not Confirmed, Current Received I-Field in Sequence

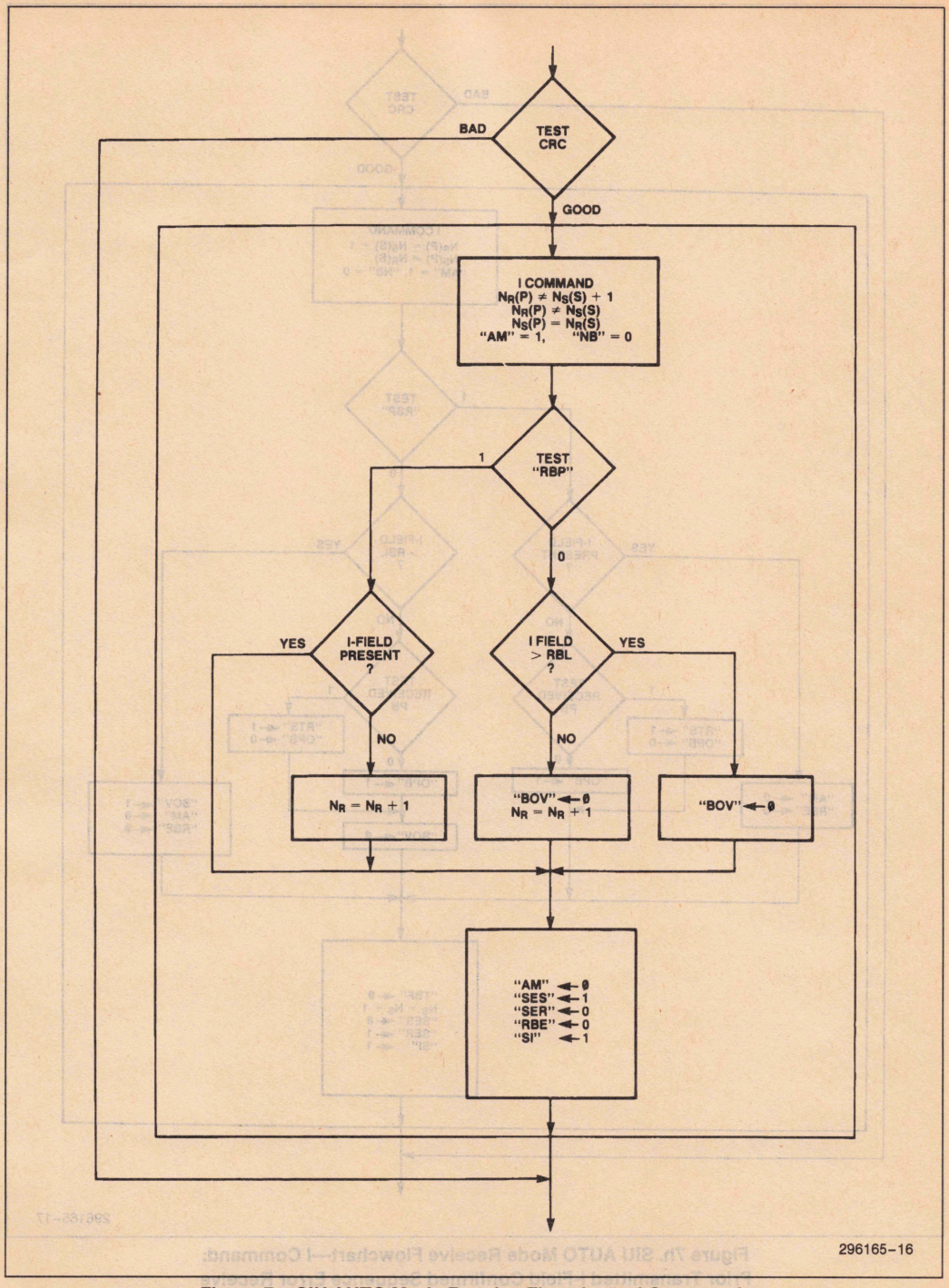


Figure 7g. SIU AUTO Mode Receive Flowchart—I Command: Sequence Error Send, Current Received I-Field in Sequence

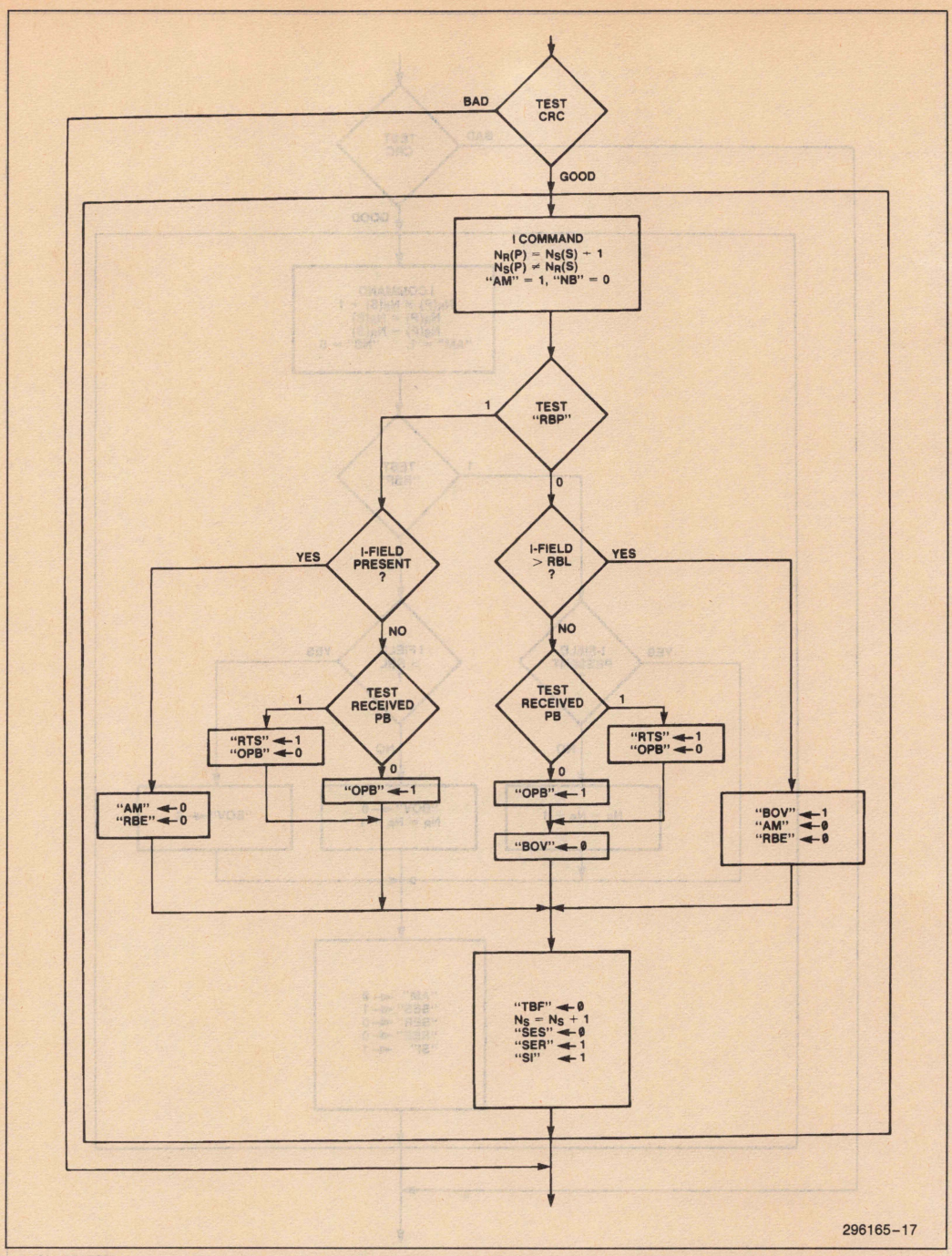


Figure 7h. SIU AUTO Mode Receive Flowchart—I Command: Prior Transmitted I-Field Confirmed Sequence Error Receive

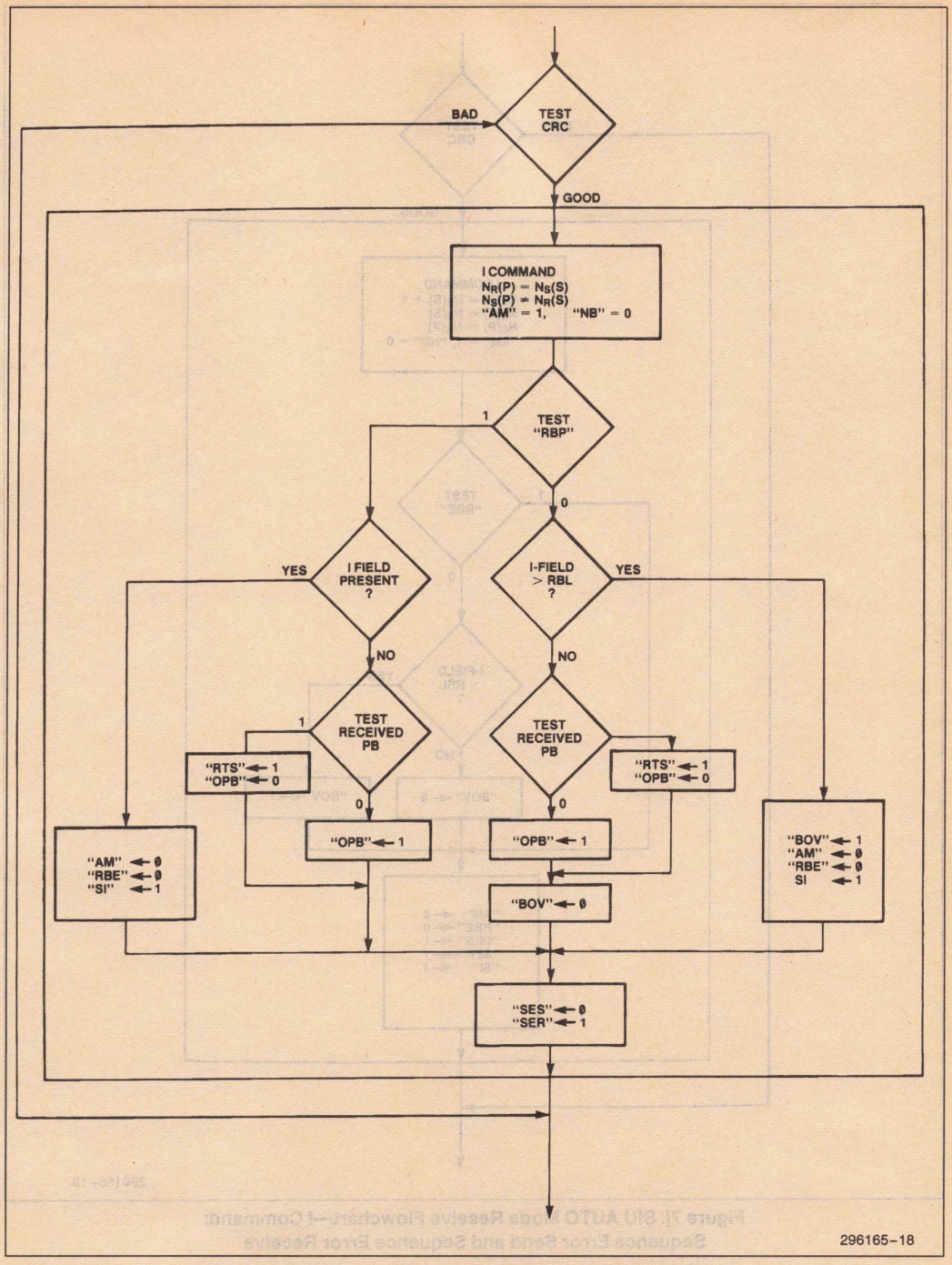


Figure 7i. SIU AUTO Mode Receive Flowchart—I Command: Prior Transmitted I-Field Not Confirmed, Sequence Error Receive

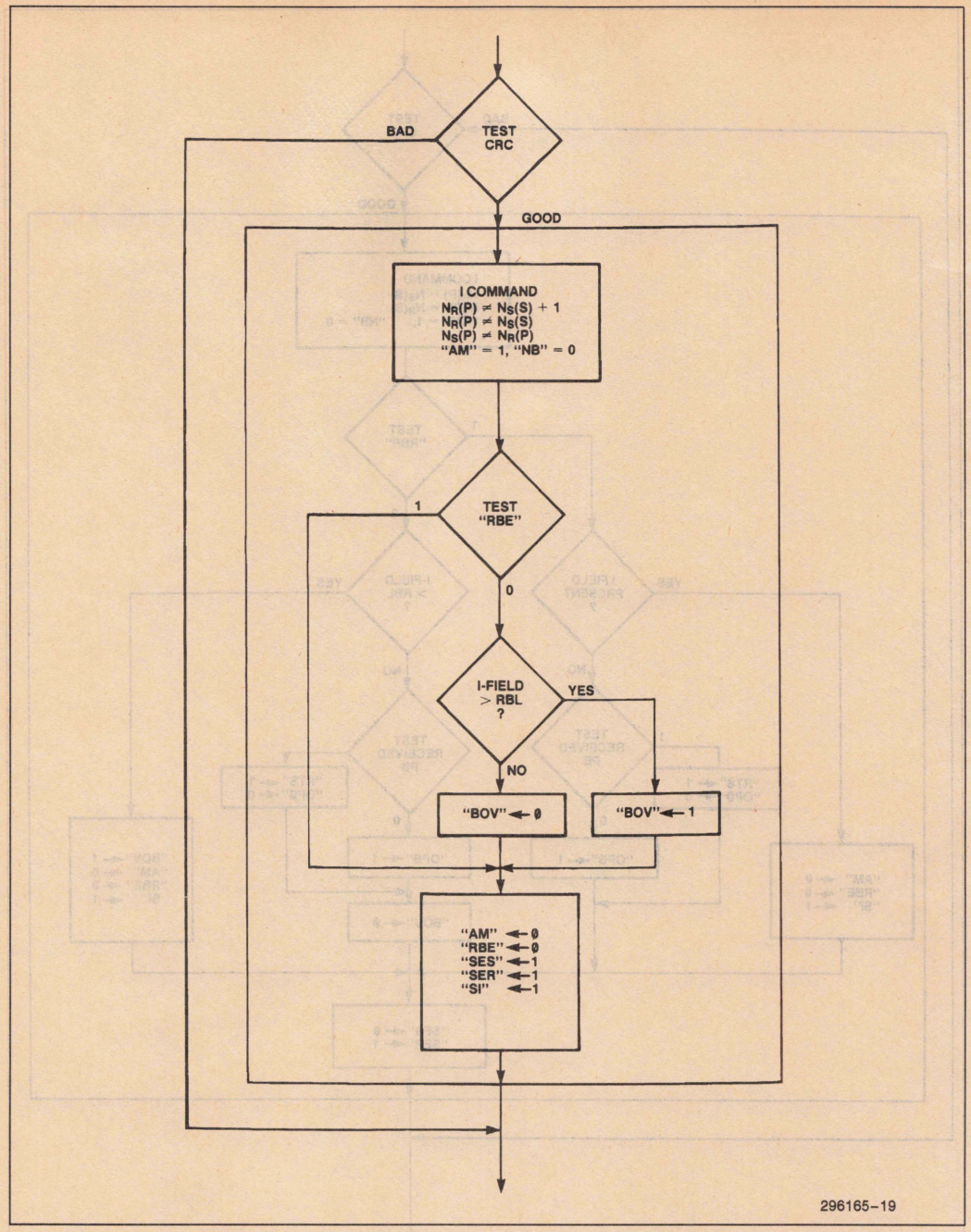


Figure 7j. SIU AUTO Mode Receive Flowchart—I Command: Sequence Error Send and Sequence Error Receive

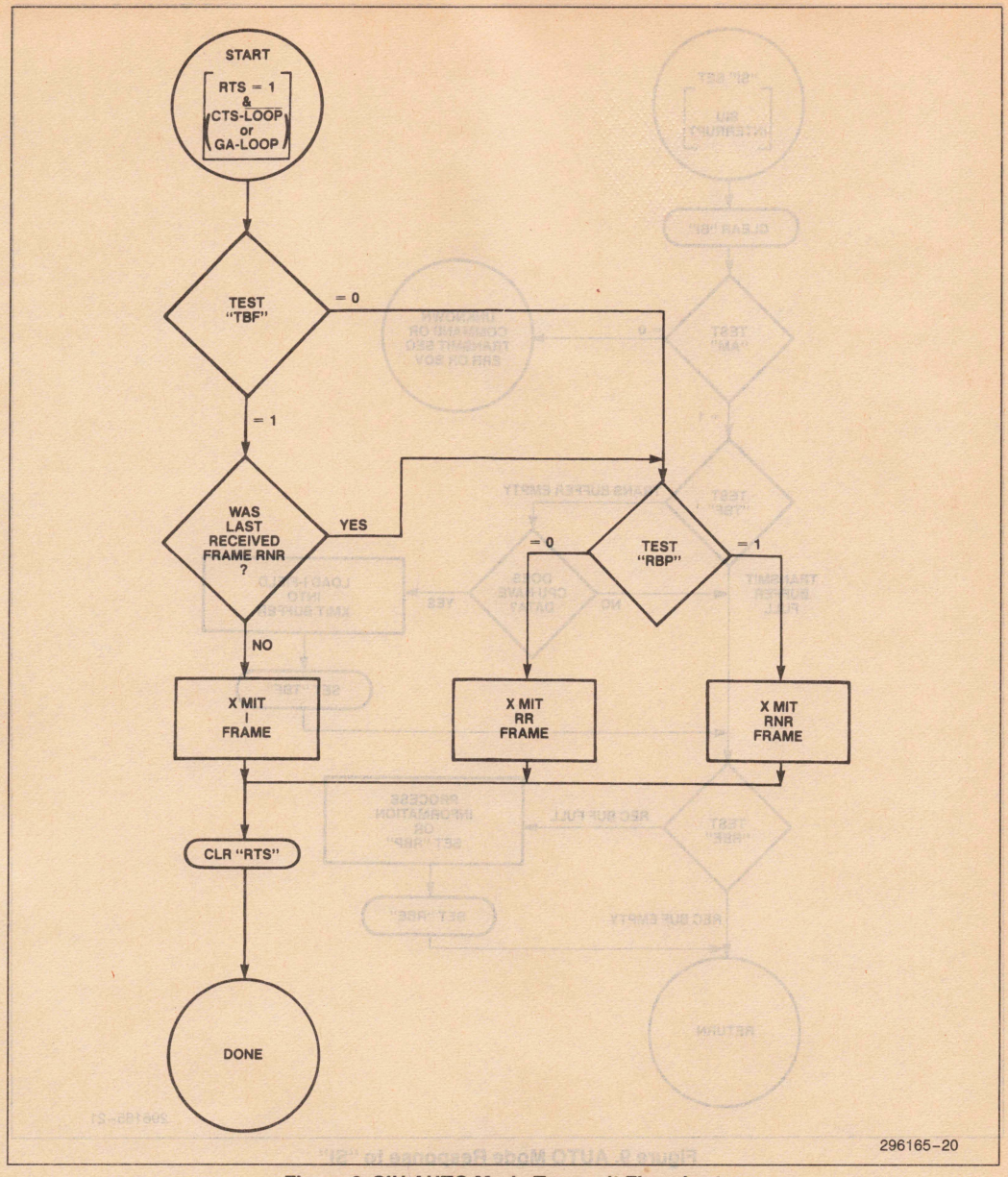


Figure 8. SIU AUTO Mode Transmit Flowchart

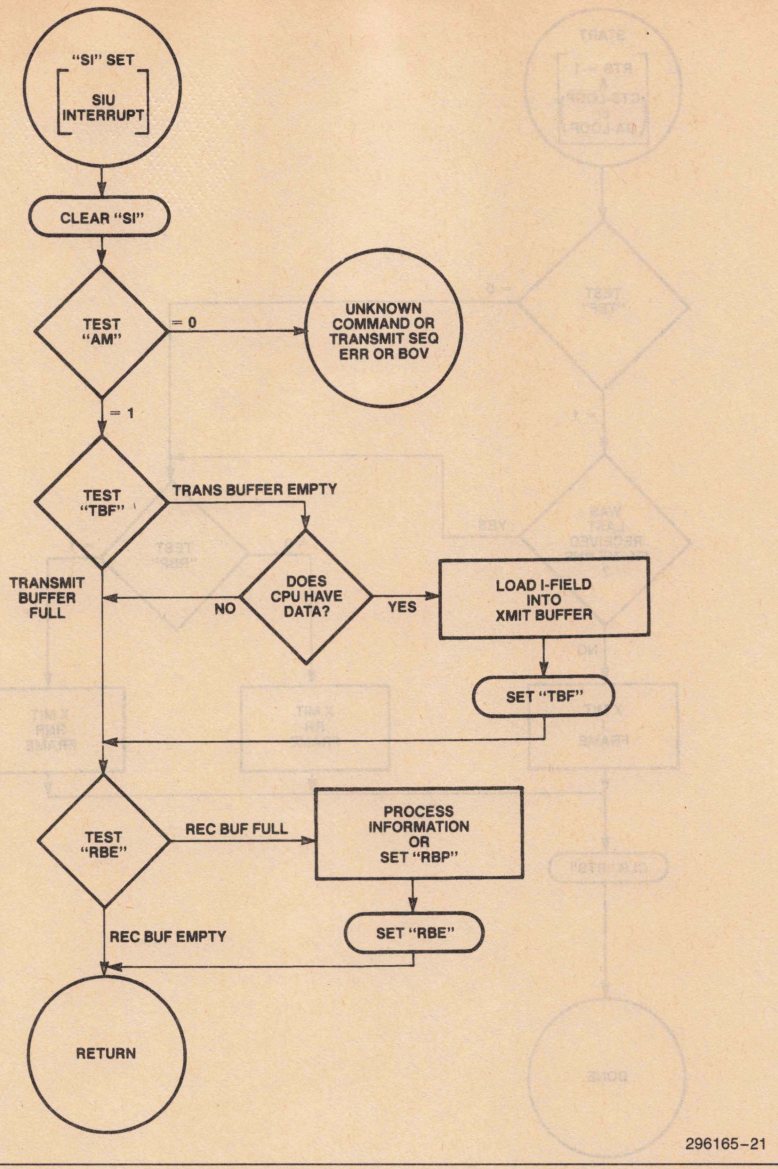


Figure 9. AUTO Mode Response to "SI"

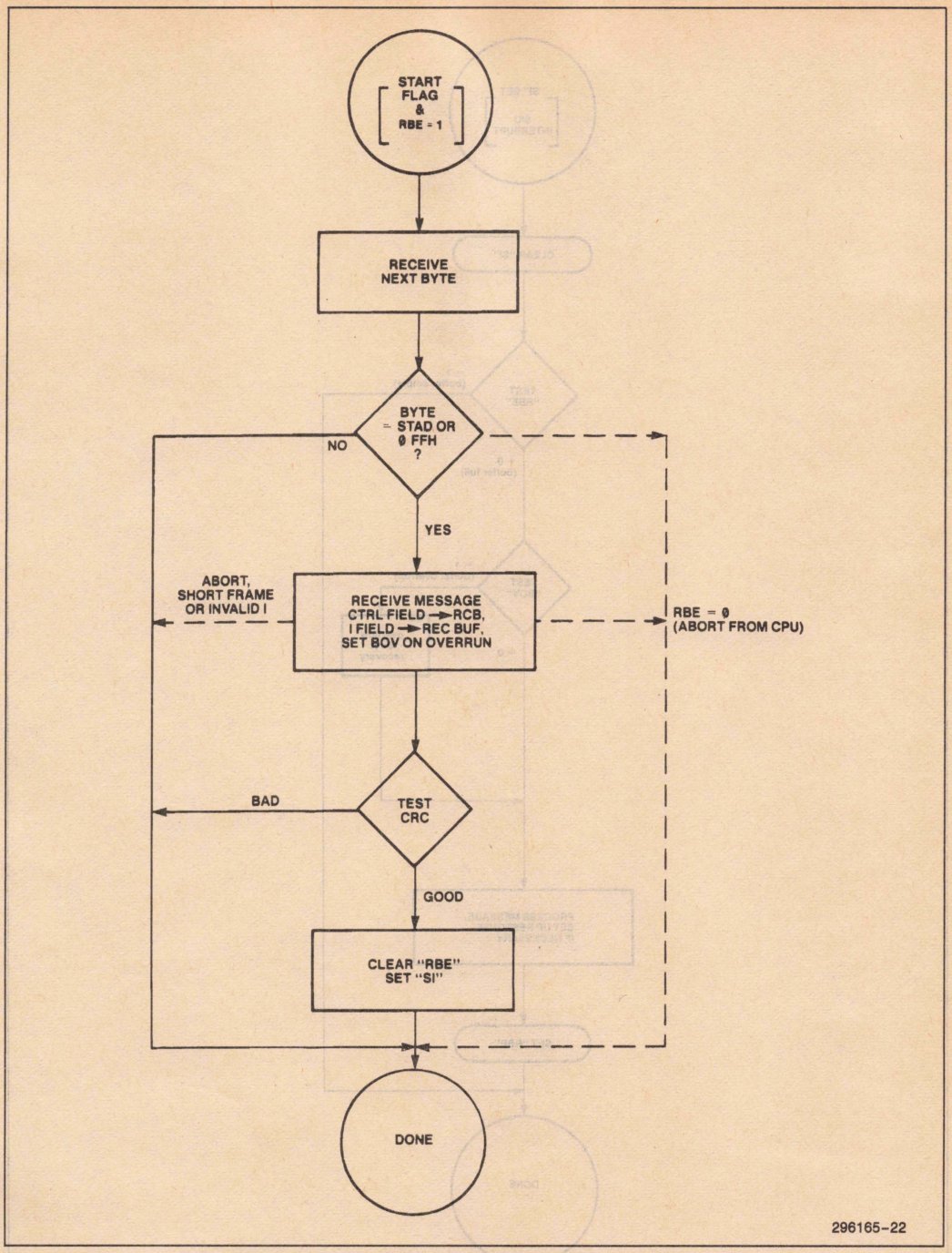


Figure 10. SIU FLEXIBLE Mode Receive Flowchart

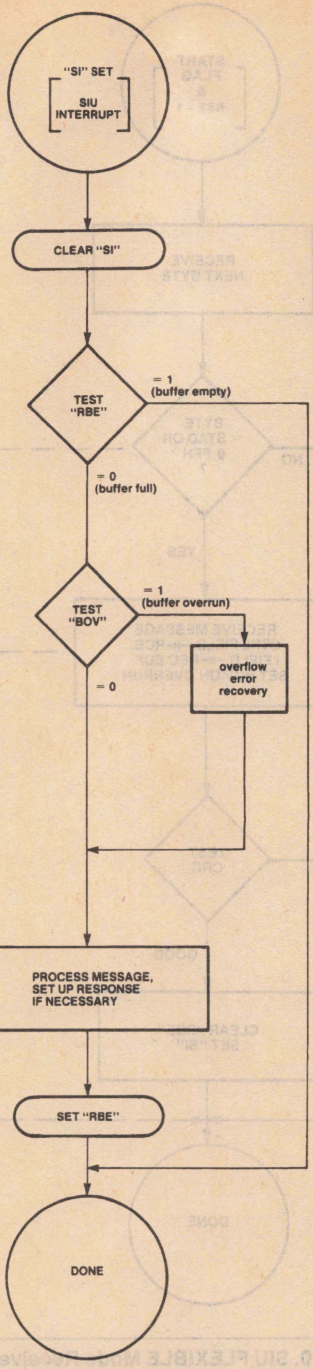


Figure 11. FLEXIBLE Mode Response to Receive "SI"

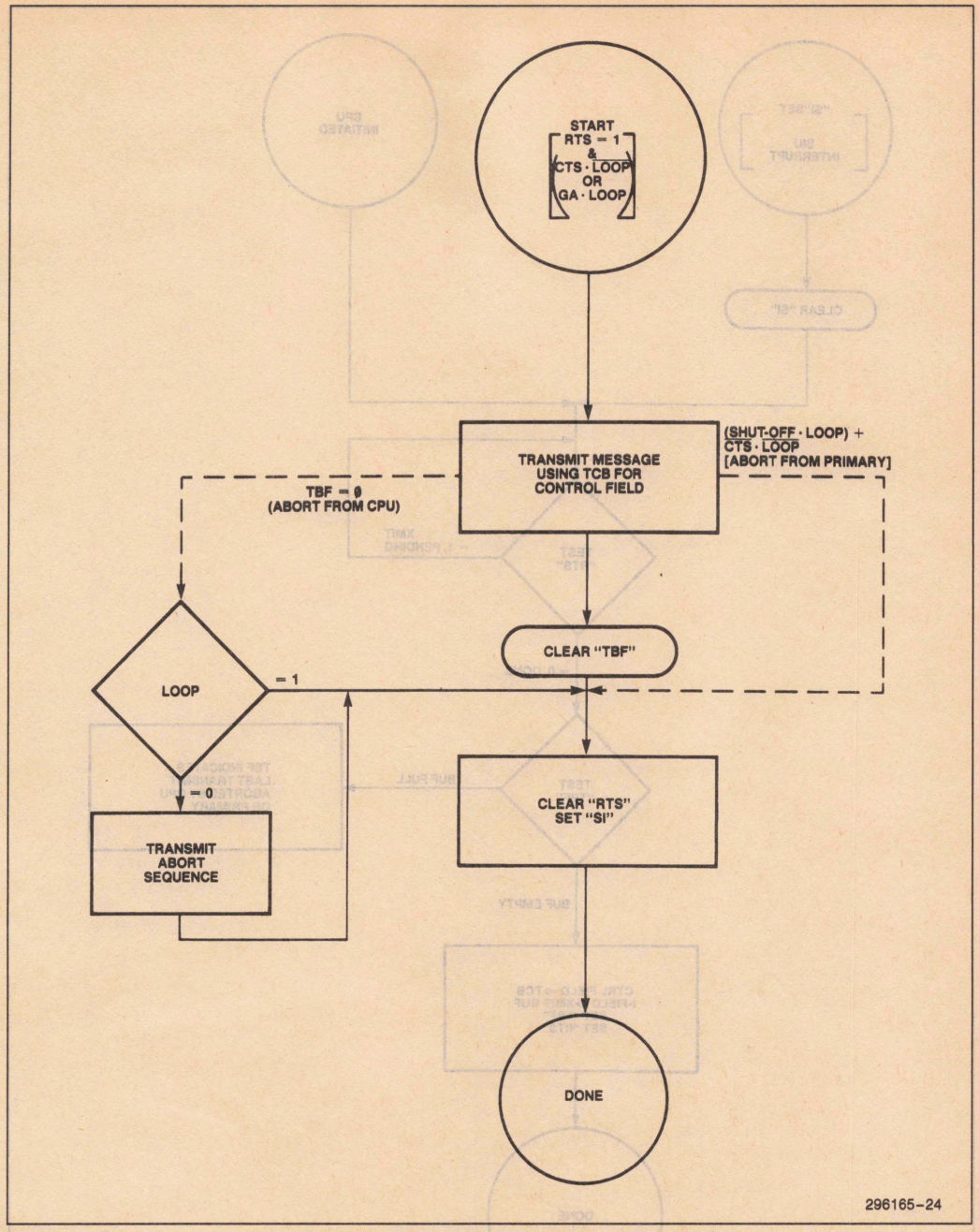


Figure 12. SIU FLEXIBLE Mode Transmit Flowchart

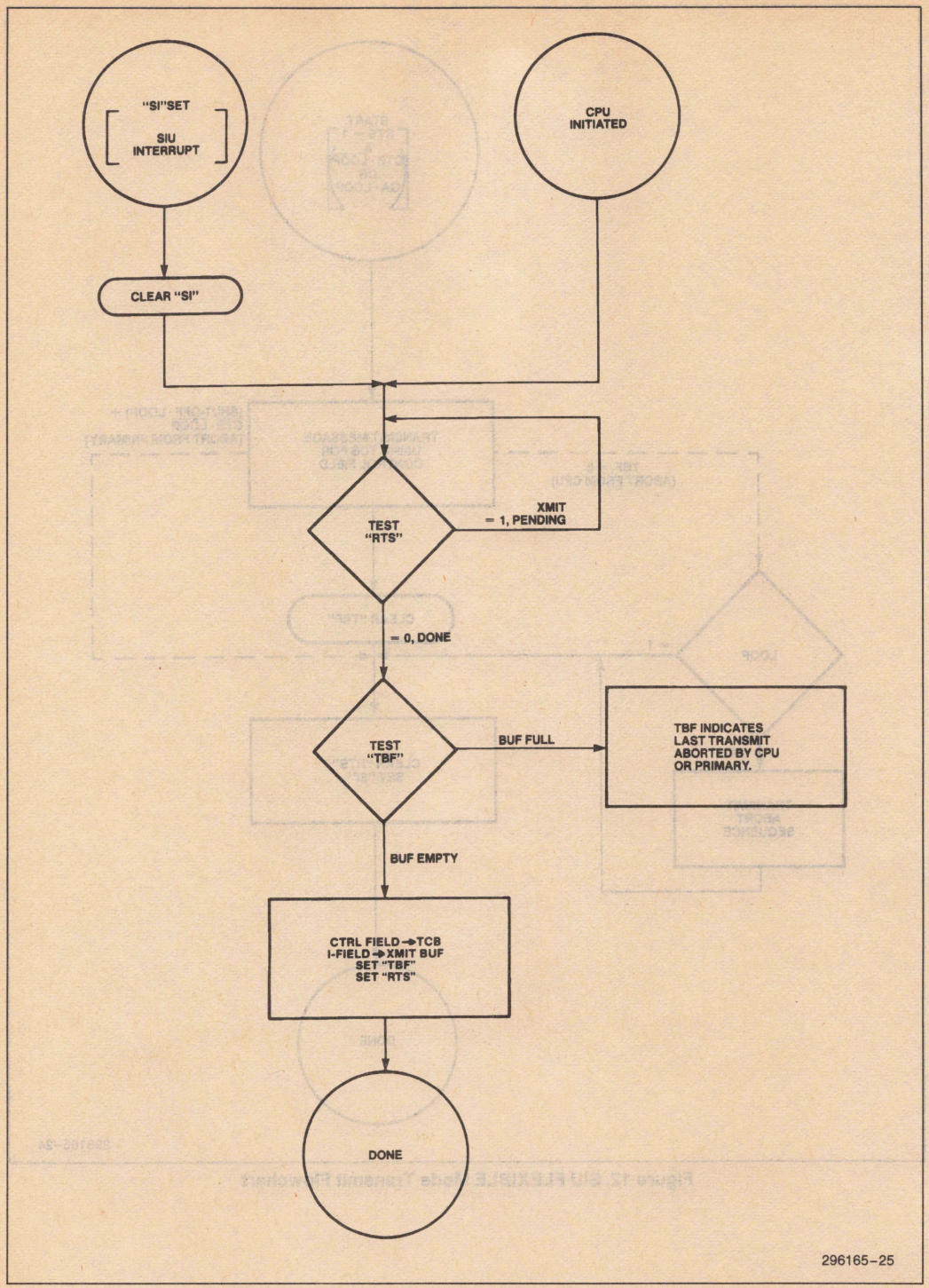


Figure 13. FLEXIBLE Mode Response to Transmit "SI"

9.0 MORE DETAILS ON SIU HARDWARE

The SIU divides functionally into two sections—a bit processor (BIP) and a byte processor (BYP)—sharing some common timing and control logic. As shown in Figure 14, the BIP operates between the serial port pins and the SIU bus, and performs all functions necessary to transmit/receive a byte of data to/from the serial data stream. These operations include shifting, NRZI encoding/decoding, zero insertion/deletion, and FCS generation/checking. The BYP manipulates bytes of data to perform message formatting, and other transmitting and receiving functions. It operates between the SIU bus (SIB) and the 8044's internal bus (IB). The interface between the SIU and the CPU involves an interrupt and some locations in on-chip RAM space which are managed by the BYP.

The maximum possible data rate for the serial port is limited to ½ the internal clock rate. This limit is imposed by both the maximum rate of DMA to the on-chip RAM, and by the requirements of synchronizing to an external clock. The internal clock rate for an 8044 running on a 12 MHz crystal is 6 MHz. Thus the maximum 8044 serial data rate is 3 MHz. This data rate drops down to 2.4 MHz when time is allowed for external clock synchronization.

9.1 The Bit Processor

In the asynchronous (self clocked) modes the clock is extracted from the data stream using the on-chip digital phase-locked-loop (DPLL). The DPLL requires a clock input at 16 times the data rate. This 16 × clock may originate from SCLK, Timer 1 Overflow, or PH2 (one half the oscillator frequency). The extra divide by-two described above allows these sources to be treated alternatively as 32 × clocks.

The DPLL is a free-running four-bit counter running off the 16 × clock. When a transition is detected in the receive data stream, a count is dropped (by suppressing the carry-in) if the current count value is greater than 8. A count is added (by injecting a carry into the second stage rather than the first) if the count is less than 8. No adjustment is made if the transition occurs at the count of 8. In this manner the counter locks in on the point at which transitions in the data stream occur at the count of 8, and a clock pulse is generated when the count overflows to 0.

In order to perform NRZI decoding, the NRZI decoder compares each bit of input data to the previous bit. There are no clock delays in going through the NRZI decoder.

The zero insert/delete circuitry (ZID) performs zero insertion/deletion, and also detects flags, GA's (Go-Ahead's), and aborts (same as GA's) in the data stream. The pattern 1111110 is detected as an early GA, so that the GA may be turned into a flag for loop mode transmission.

The shut-off detector monitors the receive data stream for a sequence of eight zeros, which is a shut-off command for loop mode transmissions. The shut-off detector is a three-bit counter which is cleared whenever a one is found in the receive data stream. Note that the ZID logic could not be used for this purpose, because the receive data must be monitored even when the ZID is being used for transmission.

As an example of the operation of the bit processor, the following sequence occurs in relation to the receive data:

1) RXD is sampled by SCLK, and then synchronized to the internal processor clock (IPC).
2) If the NRZI mode is selected, the incoming data is NRZI decoded.
3) When receiving other than the flag pattern, the ZID deletes the '0' after 5 consecutive '1's (during transmission this zero is inserted). The ZID locates the byte boundary for the rest of the circuitry. The ZID deletes the '0's by preventing the SR (shift register) from receiving a clocking pulse.
4) The FCS (which is a function of the data between the flags—not including the flags) is initialized and started at the detection of the byte boundary at the end of the opening flag. The FCS is computed each bit boundary until the closing flag is detected. Note that the received FCS has gone through the ZID during transmission.

9.2 The Byte Processor

Figure 15 is a block diagram of the byte processor (BYP). The BYP contains the registers and controllers necessary to perform the data manipulations associated with SDLC communications. The BYP registers may be read or written by the CPU over the 8044's internal bus (IB), using standard 8044 hardware register operations. The 8044 register select PLA controls these operations. Three of the BYP registers connect to the IB through the IBS, a sub-bus which also connects to the CPU interrupt control registers.

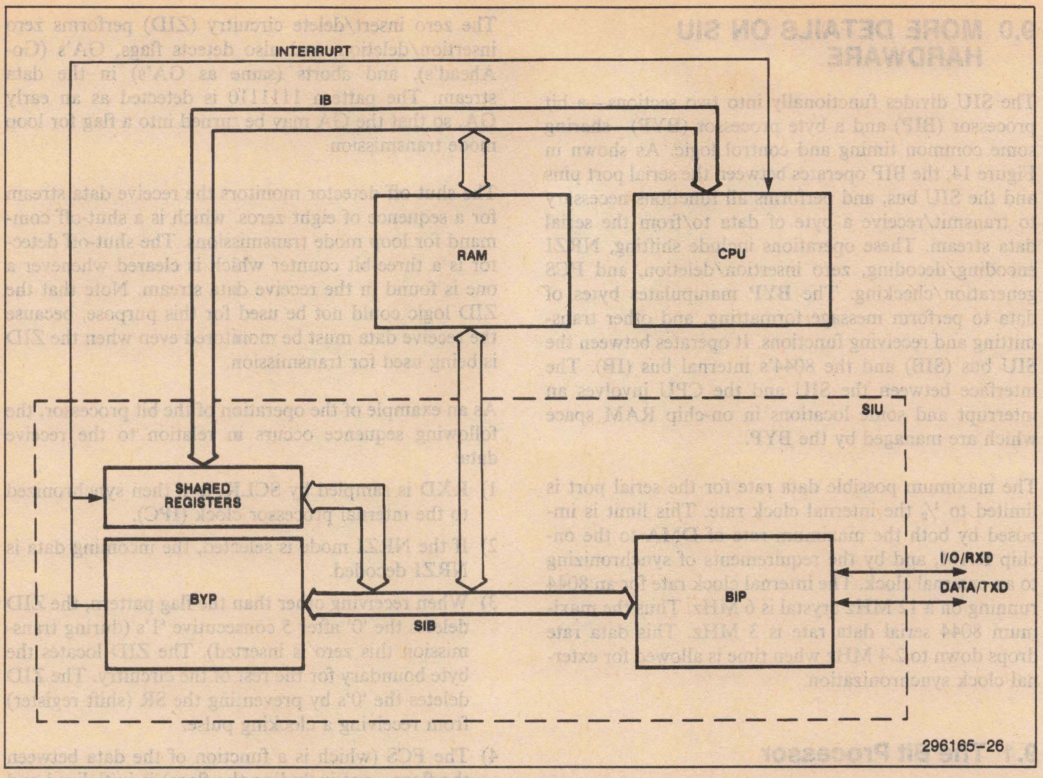


Figure 14. The Bit and Byte Processors

Simultaneous access of a register by both the IB and the SIB is prevented by timing. In particular, RAM access is restricted to alternate internal processor cycles for the CPU and the SIU, in such a way that collisions do not occur.

As an example of the operation of the byte processor, the following sequence occurs in relation to the receive data:

1) Assuming that there is an address field in the frame, the BYP takes the station address from the register file into temporary storage. After the opening flag, the next field (the address field) is compared to the station address in the temporary storage. If a match occurs, the operation continues.
2) Assuming that there is a control field in the frame, the BYP takes the next byte and loads it into the RCB register. The RCB register has the logic to update the NSNR register (increment receive count, set SES and SER flags, etc.).
3) Assuming that there is an information field, the next byte is dumped into RAM at the RBS location. The DMA CNT (RBL at the opening flag) is loaded from the DMA CNT register into the RB register and decremented. The RFL is then loaded into the RB register, incremented, and stored back into the register file.
4) This process continues until the DMA CNT reaches zero, or until a closing flag is received. Upon either event, the BYP updates the status, and, if the CRC is good, the NSNR register.

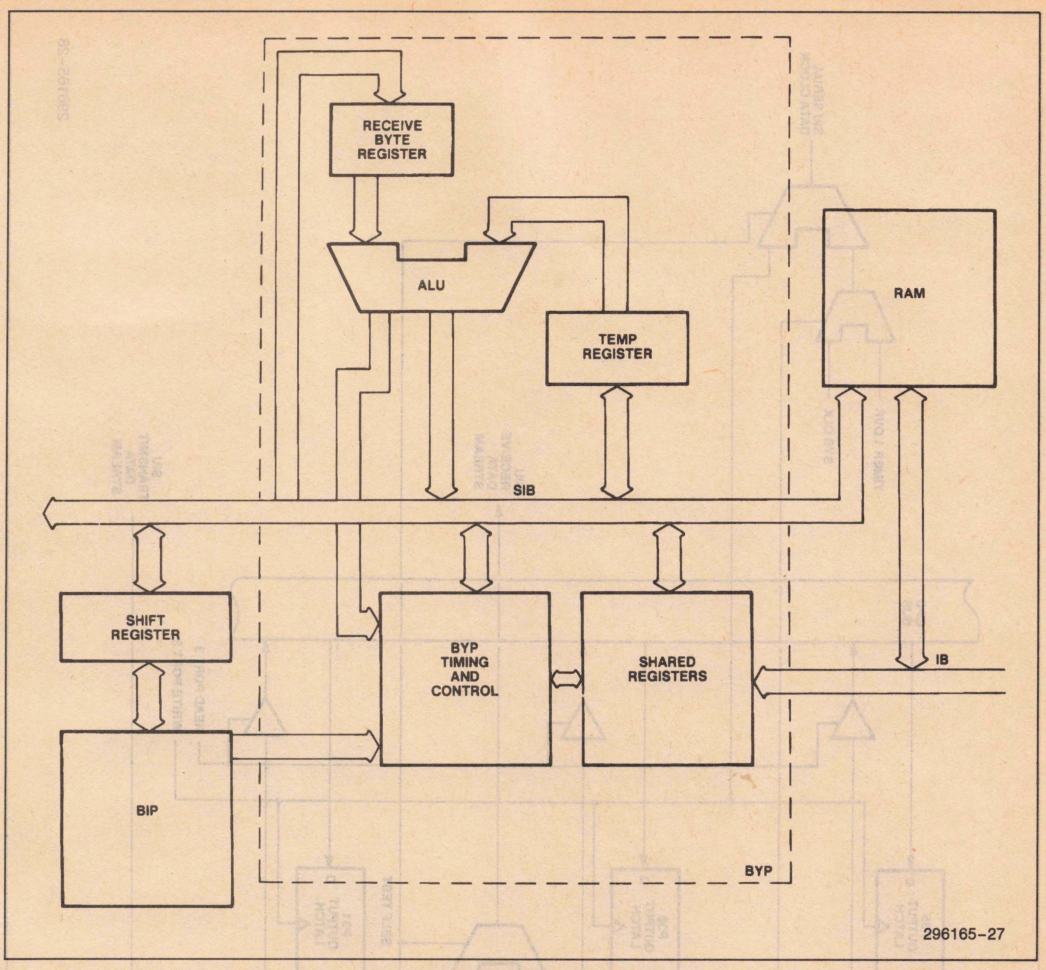


Figure 15. The Byte Processor

10.0 DIAGNOSTICS

An SIU test mode has been provided, so that the on-chip CPU can perform limited diagnostics on the SIU. The test mode utilizes the output latches for P3.0 and P3.1 (pins 10 and 11). These port 3 pins are not useful as out-put ports, since the pins are taken up by the serial port functions. Figure 16 shows the signal routing associated with the SIU test mode.

Writing a 0 to P3.1 enables the serial test mode (P3.1 is set to 1 by reset). In test mode the P3.0 bit is mapped into the received data stream, and the 'write port 3' control signal is mapped into the SCLK path in place of T1. Thus, in test mode, the CPU can send a serial data stream to the SIU by writing to P3.0. The transmit data stream can be monitored by reading P3.1. Each successive bit is transmitted from the SIU by writing to any bit in Port 3, which generates SCLK.

In test mode, the P3.0 and P3.1 pins are placed in a high voltage, high impedance state. When the CPU reads P3.0 and P3.1 the logic level applied to the pin will be returned. In the test mode, when the CPU reads 3.1, the transmit data value will be returned, not the voltage on the pin. The transmit data remains constant for a bit time. Writing to P3.0 will result in the signal being outputted for a short period of time. However, since the signal is not latched, P3.0 will quickly return to a high voltage, high impedance state.

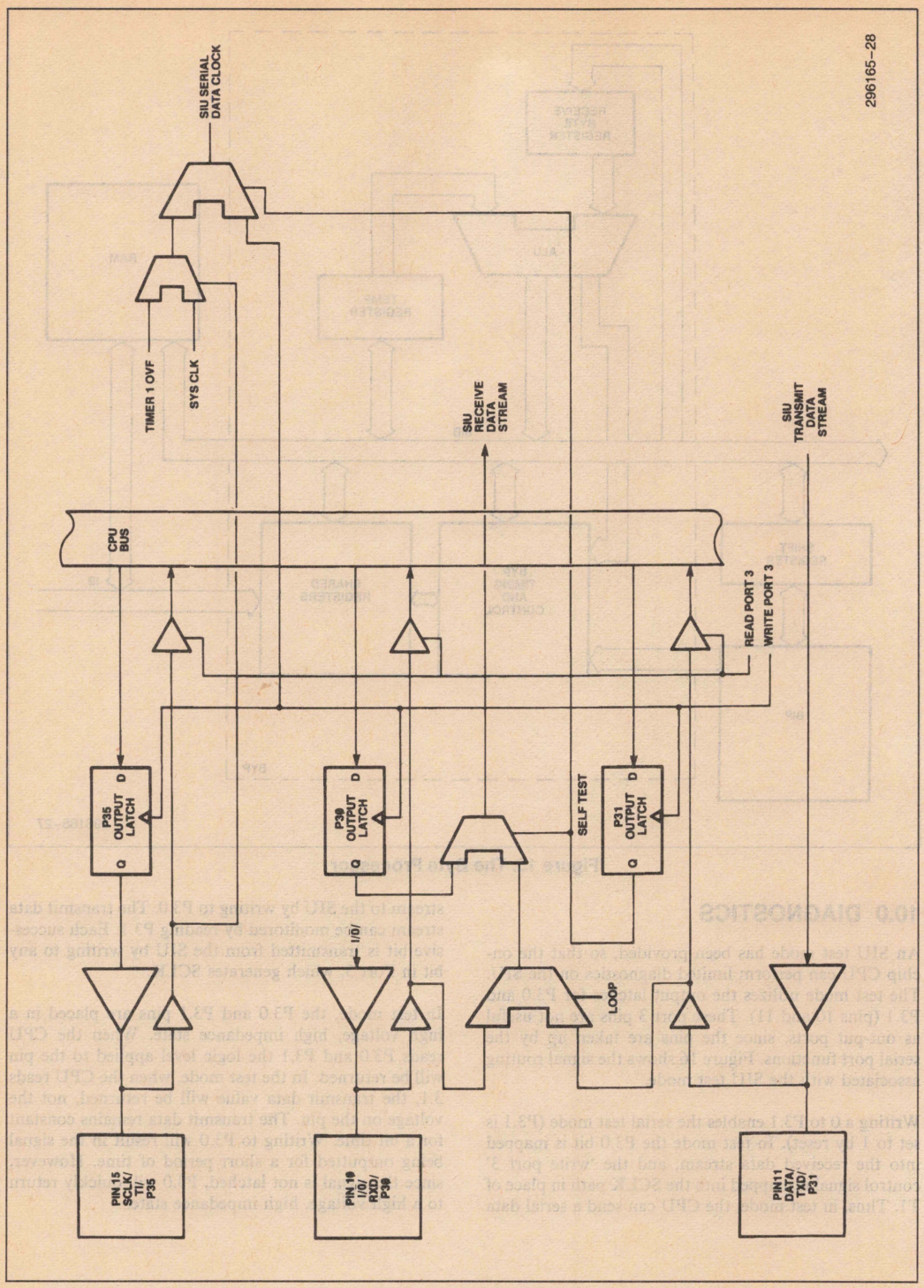


Figure 16. SIU Test Mode

The serial test mode is disabled by writing a 1 to P3.1. Care must be taken that a 0 is never written to P3.1 in the course of normal operation, since this causes the test mode to be entered.

Figure 17 is an example of a simple program segment that can be imbedded into the user's diagnostic program. That example shows how to put the 8044 into "Loop-back mode" to test the basic transmitting and receiving functions of the SIU.

Loop-back mode is functionally equivalent to a hardwire connection between pins 10 and 11 on the 8044.

In this example, the 8044 CPU plays the role of the primary station. The SIU is in the AUTO mode. The CPU sends the SIU a supervisory frame with the poll bit set and an RNR command. The SIU responds with a supervisory frame with the poll bit set and an RR command.

The operation proceeds as follows:

Interrupts are disabled, and the self test mode is enabled by writing a zero to P3.1. This establishes P3.0 as the data path from the CPU to the SIU. CTS (clear-to-send) is enabled by writing a zero to P1.7. The station address is initialized by writing 08AH into the STAD (station address register).

The SIU is configured for receive operation in the clocked mode and in AUTO mode. The CPU then transmits a supervisory frame. This frame consists of an opening flag, followed by the station address, a control field indicating that this is a supervisory frame with an RNR command, and then a closing flag.

Each byte of the frame is transmitted by writing that byte into the A register and then calling the subroutine XMIT8. Two additional SCLKs are generated to guarantee that the last bits in the frame have been clocked into the SIU. Finally the CPU reads the status register (STS). If the operation has proceeded correctly, the status will be 072H. If it is not, the program jumps to the ERROR loop and terminates.

The SIU generates an SI (SIU interrupt) to indicate that it has received a frame. The CPU clears this interrupt, and then begins to monitor the data stream that is being generated by the SIU in response to what it has received. As each bit arrives (via P3.1), it is moved into the accumulator, and the CPU compares the byte in the accumulator with 07EH, which is the opening flag. When a match occurs, the CPU identifies this as byte boundary, and thereafter processes the information byte-to-byte.

The CPU calls the RCV8 subroutine to get each byte into the accumulator. The CPU performs compare operations on (successively) the station address, the control field (which contains the RR response), and the closing flag. If any of these do not compare, the program jumps to the ERROR loop. If no error is found, the program jumps to the DONE loop.

```
MCS-51 MACRO ASSEMBLER    DATA

ISIS-II MCS-51 MACRO ASSEMBLER V2.0
OBJECT MODULE PLACED IN :F1:DATA.OBJ
ASSEMBLER INVOKED BY:  asm51 :f1:data.man device(44)


LOC  OBJ             LINE     SOURCE

                        1
                        2
0000 75C800             3     INIT:   MOV     STS,#00H
0003 C2B1               4             CLR     P3.1            ; Enable self test mode
0005 C297               5             CLR     P1.7            ; Enable CTS
0007 75CE8A             6             MOV     STAD, #8AH      ; Initialize address
                        7
                        8       ; CONFIGURE RECEIVE OPERATION
                        9
000A 75DB6A            10             MOV     NSNR, #6AH      ; NS(S)=3, SES=0, NR(S)=5, SER=0
000D 75C901            11             MOV     SMD, #01H       ; NFCS=1
0010 75C8C2            12             MOV     STS, #0C2H      ; TBF=1, RBE=1, AM=1
                       13
                       14     ;   TRANSMIT A SUPERVISORY FRAME FROM THE PRIMARY STATION WITH THE POLL
                       15     ;   BIT SET AND A RNR COMMAND
                       16
0013 747E              17     SEND:   MOV     A, #7EH         ; The SIU receives a flag first
0015 120066            18             CALL    XMITB
0018 748A              19             MOV     A, #8AH         ; The address is next
001A 120066            20             CALL    XMITB
001D 7495              21             MOV     A, #095H        ; RNR SUP FRAME with P/F=1, NR(P)=4
001F 120066            22             CALL    XMITB
0022 747E              23             MOV     A, #7EH         ; Receive closing flag
0024 120066            24             CALL    XMITB
0027 D2B0              25             SETB    P3.0            ; Generate extra SCLK's to
0029 D2B0              26             SETB    P3.0            ; Initiate receive action
                       27
002B E5C8              28             MOV     A, STS          ; Check for appropriate status
002D B4722A            29             CJNE    A, #72H, ERROR
                       30
                       31     ; PREPARE TO RECEIVE RUPI'S RESPONCE TO PRIMARY'S RNR
                       32
                       33
                       34
0030 C2CC              35     RECV:   CLR     SI              ; Clear SI
0032 7400              36             MOV     A, #00H         ; Clear ACC
0034 780C              37             MOV     R3, #12         ; Try 12 times
                       38
                       39     ;  LOOK FOR THE OPENING FLAG
                       40
0036 D2B0              41     WFLAG1: SETB    P3.0            ; SCLK
0038 A2B1              42             MOV     C, P3.1         ; Transmitted data
003A 13                43             RRC     A
003B B47E03            44             CJNE    A, #07EH, WFLG1
003E 020046            45             JMP     CNTINU
0041 DBF3              46     WFLG1:  DJNZ    R3, WFLAG1
0043 02005A            47             JMP     ERROR
                       48
                       49
0046 12005C            50     CNTINU: CALL    RCVB            ; Get SIU's Transmitted address field
0049 B48A0E            51             CJNE    A, #08AH, ERROR
004C 12005C            52             CALL    RCVB            ; Primary expects to receive RR from SIU
004F B4B108            53             CJNE    A, #0B1H, ERROR
0052 12005C            54             CALL    RCVB            ; Receive closing flag
0055 B47E02            55             CJNE    A, #07EH, ERROR
                       56
0058 80FE              57     DONE:   JMP     DONE
                       58
005A 80FE              59     ERROR:  JMP     ERROR
                       60
                       61
005C 7808              62       RCVB: MOV     R0, #08         ; Initialize the bit counter
005E D2B0              63     GETBIT: SETB    P3.0            ; SCLK
0060 A2B1              64             MOV     C, P3.1         ; Transmitted data
0062 13                65             RRC     A
0063 DBF9              66             DJNZ    R0, GETBIT
0065 22                67             RET
                       68
                       69
                       70
0066 7809              71     XMITB:  MOV     R0, #9          ; Initialize the bit counter
0068 13                72        L3:  RRC     A               ; Put the bit to be transmitted
                       73                                     ; in the Carry
0069 D801              74             DJNZ    R0, L1          ; When all bits have been sent
006B 22                75             RET                     ; return
                       76
006C 4004              77        L1:  JC      L2              ; If the carry bit is set, set
                       78                                     ; port P3.0 else
006E C2B0              79             CLR     P3.0            ; clear port P3.0
0070 80F6              80             JMP     L3
                       81
0072 D2B0              82        L2:  SETB    P3.0
0074 80F2              83             JMP     L3
                       84     end
```

296165-29

Figure 17. Loop-Back Mode Software

8044 Application Examples 22

8044 APPLICATION EXAMPLES

1.0 INTERFACING THE 8044 TO A MICROPROCESSOR

The 8044 is designed to serve as an intelligent controller for remote peripherals. However, it can also be used as an intelligent HDLC/SDLC front end for a microprocessor, capable of extensively off-loading link control functions for the CPU. In some applications, the 8044 can even be used for communications preprocessing, in addition to data link control.

This section describes a sample hardware interface for attaching the 8044 to an 8088. It is general enough to be extended to other microprocessors such as the 8086 or the 80186.

OVERVIEW

A sample interface is shown in Figure 1. Transmission occurs when the 8088 loads a 64 byte block of memory with some known data. The 8088 then enables the 8237A to DMA this data to the 8044. When the 8044 has received all of the data from the 8237A, it sends the data in a SDLC frame. The frame is captured by the Spectron Datascope™* which displays it on a CRT in hex format.

In reception, the Datascope sends a SDLC information frame to the 8044. The 8044 receives the SDLC frame, buffers it, and sends it to the 8088's memory. In this example the 8044 is being operated in the NON-AUTO mode; therefore, it does not need to be polled by a primary station in order to transmit.

THE INTERFACE

The 8044 does not have a parallel slave port. The 8044's 32 I/O lines can be configured as a local microprocessor bus master. In this configuration, the 8044 can expand the ROM and RAM memory, control peripherals, and communicate with a microprocessor.

The 8044, like the 8051, does not have a Ready line, so there is no way to put the 8044 in wait state. The clock on the 8044 cannot be stopped. Dual port RAM could still be used, however, software arbitration would be the only way to prevent collisions. Another way to interface the 8044 with another CPU is to put a FIFO or queue between the two processors, and this was the method chosen for this design.

Figure 2 shows the schematic of the 8044/8088 interface. It involves two 8-bit tri-state latches, two SR flip-flops, and some logic gates (6 TTL packs). The circuitry implements a one byte FIFO. RS422 transceivers are used, which can be connected to a multidrop link. Figure 3 shows the 8088 and support circuitry; the memory and decoders are not shown. It is a basic 8088 Min Mode system with an 8237A DMA controller and an 8259A interrupt controller.

DMA Channel One transfers a block of memory to the tri-state latch, while Channel Zero transfers a block of data from the latch to 8088's memory. The 8044's Interrupt 0 signal vectors the CPU into a routine which reads from the internal RAM and writes to the latch. The 8044's Interrupt 1 signal causes the chip to read from the latch and write to its on-chip data RAM. Both DMA requests and acknowledges are active low.

Initially, when the power is applied, a reset pulse coming from the 8284A initializes the SR flip-flops. In this initialization state, the 8044's transmit interrupt and the 8088's transmit DMA request are active; however, the software keeps these signals disabled until either of the two processors are ready to transmit. The software leaves the receive signals enabled, unless the receive buffers are full. In this way either the 8088 or the 8044 are always ready to receive, but they must enable the transmit signal when they have prepared a block to transmit. After a block has been transmitted or received, the DMA and interrupt signals return to the initial state.

The receive and transmit buffer sizes for the blocks of data sent between the 8044 and the 8088 have a maximum fixed length. In this case the buffer size was 64 bytes. The buffer size must be less than 192 bytes to enable 8044 to buffer the data in its on-chip RAM. This design allows blocks of data that are less than 64 bytes, and accommodates networks that allow frames of varying size. The first byte transferred between the 8088 and the 8044 is the byte count to follow; thus the 8044 knows how many bytes to receive before it transmits the SDLC frame. However, when the 8044 sends data to the 8088's memory, the 8237A will not know if the 8044 will send less than the count the 8237A was programmed for. To solve this problem, the 8237A is operated in the single mode. The 8044 uses an I/O bit to generate an interrupt request to the 8259A. In the 8088's interrupt routine, the 8237A's receive DMA channel is disabled, thus allowing blocks of data less than 64 bytes to be received.

THE SOFTWARE

The software for the 8044 and the 8088 is shown in Table 1. The 8088 software was written in PL/M86, and the 8044 software was written in assembly language.

The 8044 software begins by initializing the stack, interrupt priorities, and triggering types for the interrupts. At this point, the SIU parameter registers are

*Datascope is a trademark of Spectron Inc.

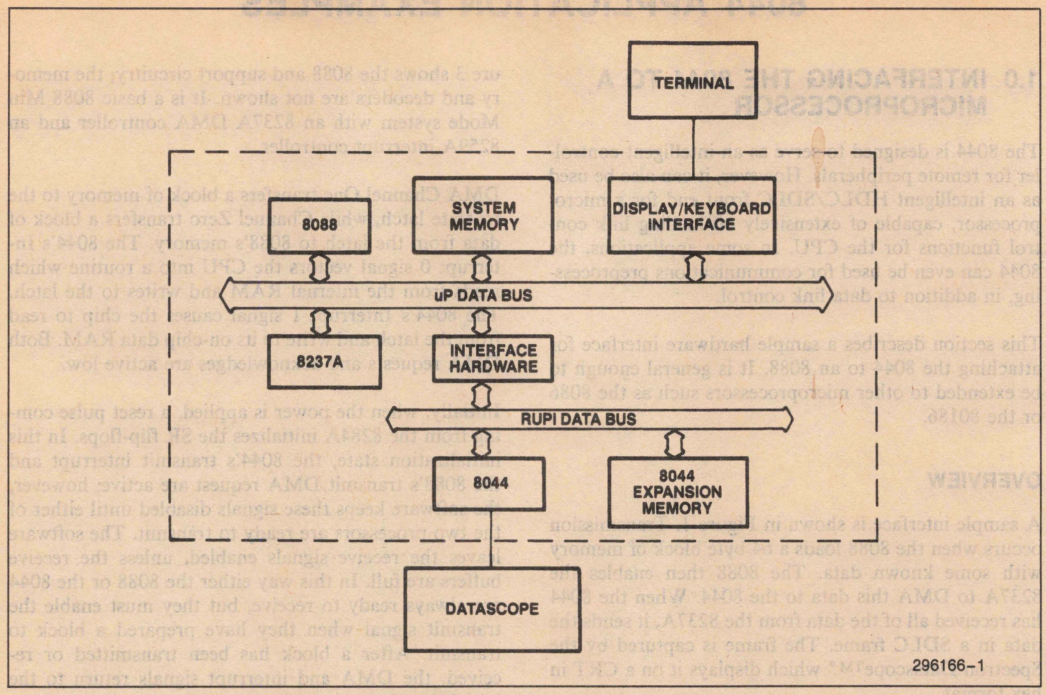


Figure 1. Block Diagram of 8088/8044 Interface Test

initialized. The receive and transmit buffer starting addresses and lengths are loaded for the on-chip DMA. This DMA is for the serial port. The serial station address and the transmit control bytes are loaded too.

Once the initialization has taken place, the SIU interrupt is enabled, and the external interrupt which receives bytes from the 8088 is enabled. Setting the 8044's Receive Buffer Empty (RBE) bit enables the receiver. If this bit is reset, no serial data can be received. The 8044 then waits in a loop for either RECEIVE DMA interrupt or the SERIAL INT interrupt.

The RECEIVE DMA interrupt occurs when the 8237A is transferring a block of data to the 8044. The first time this interrupt occurs, the 8044 reads the latch and loads the count value into the R2 register. On subsequent interrupts, the 8044 reads the latch, loads the data into the transmit buffer, and decrements R2. When R2 reaches zero, the interrupt routine sends the data in an SDLC frame, and disables the RECEIVE DMA interrupt. After the frame has been transmitted, a serial interrupt is generated. The SERIAL INT routine detects that a frame has been transmitted and re-enables the RECEIVE DMA interrupt. Thus, while the frame is being transmitted through the SIU, the 8237A is inhibited from sending data to the 8044's transmit buffer.

The TRANSMIT DMA routine sends a block of data from the 8044's receive buffer to the 8088's memory. Normally this interrupt remains disabled. However, if a serial interrupt occurs, and the SERIAL INT routine detects that a frame has been received, it calls the SEND subroutine. The SEND subroutine loads the number of bytes which were received in the frame into the receive buffer. Register R1 points to the receive buffer and R2 is loaded with the count. The TRANSMIT DMA interrupt is enabled, and immediately upon returning from the SERIAL INT routine, the interrupt is acknowledged. Each time the TRANSMIT DMA interrupt occurs, a byte is read from the receive buffer, written to the latch, and R2 is decremented. When R2 reaches 0, the TRANSMIT DMA interrupt is disabled, the SIU receiver is re-enabled, and the 8044 interrupts the 8088.

CONCLUSION

For the software shown in Table 1, the transfer rate from the 8088's memory to the 8044 was measured at 75K bytes/sec. This transfer rate largely depends upon the number of instructions in the 8044's interrupt service routine. Fewer instructions result in a higher transfer rate.

There are many ways of interfacing the 8044 locally to another microprocessor: FIFO's, dual port RAM with software arbitration, and 8255's are just a few. Alternative approaches, which may be more optimal for certain applications, are certainly possible.

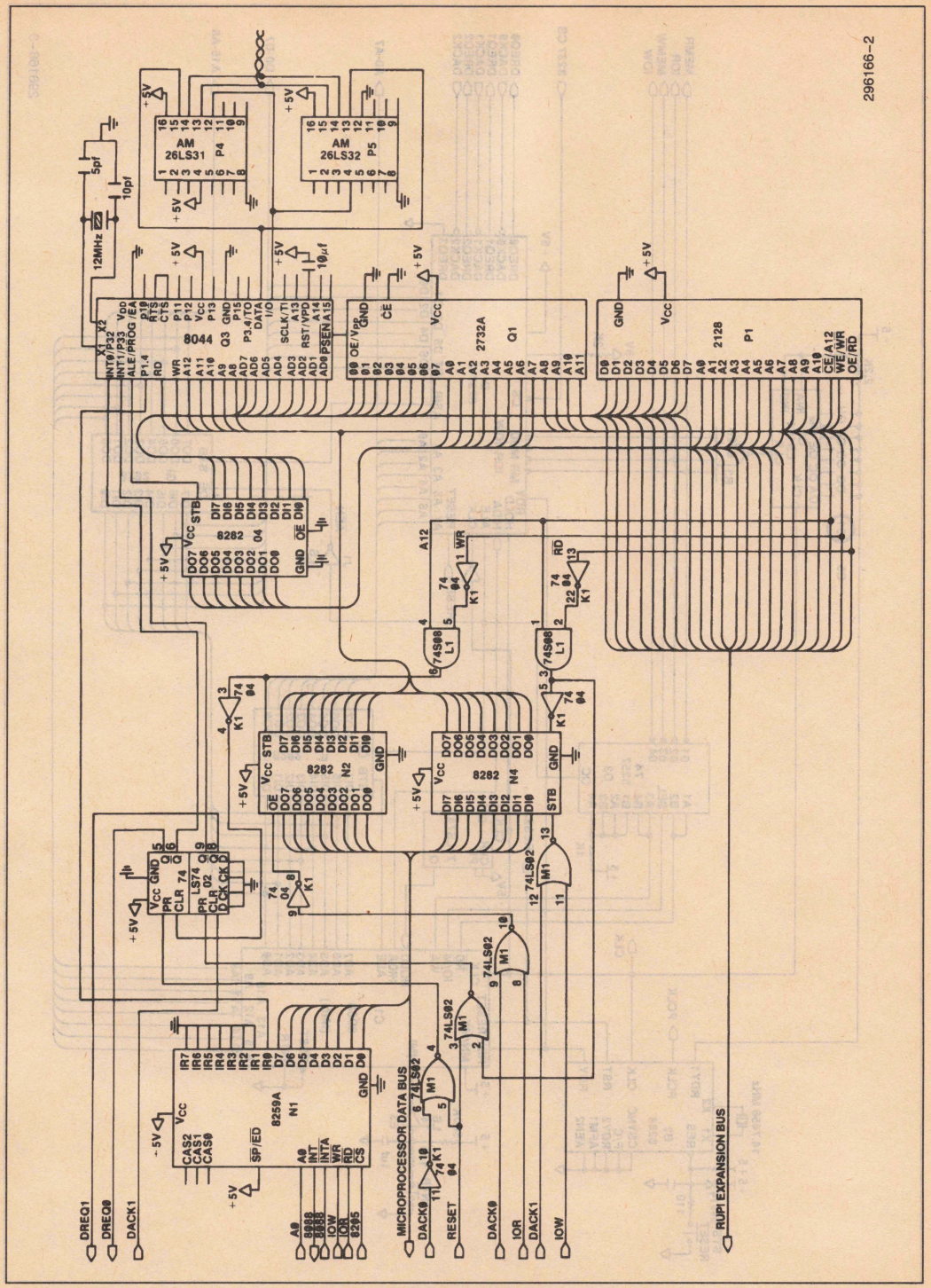


Figure 2. 8044 Interface to the 8088

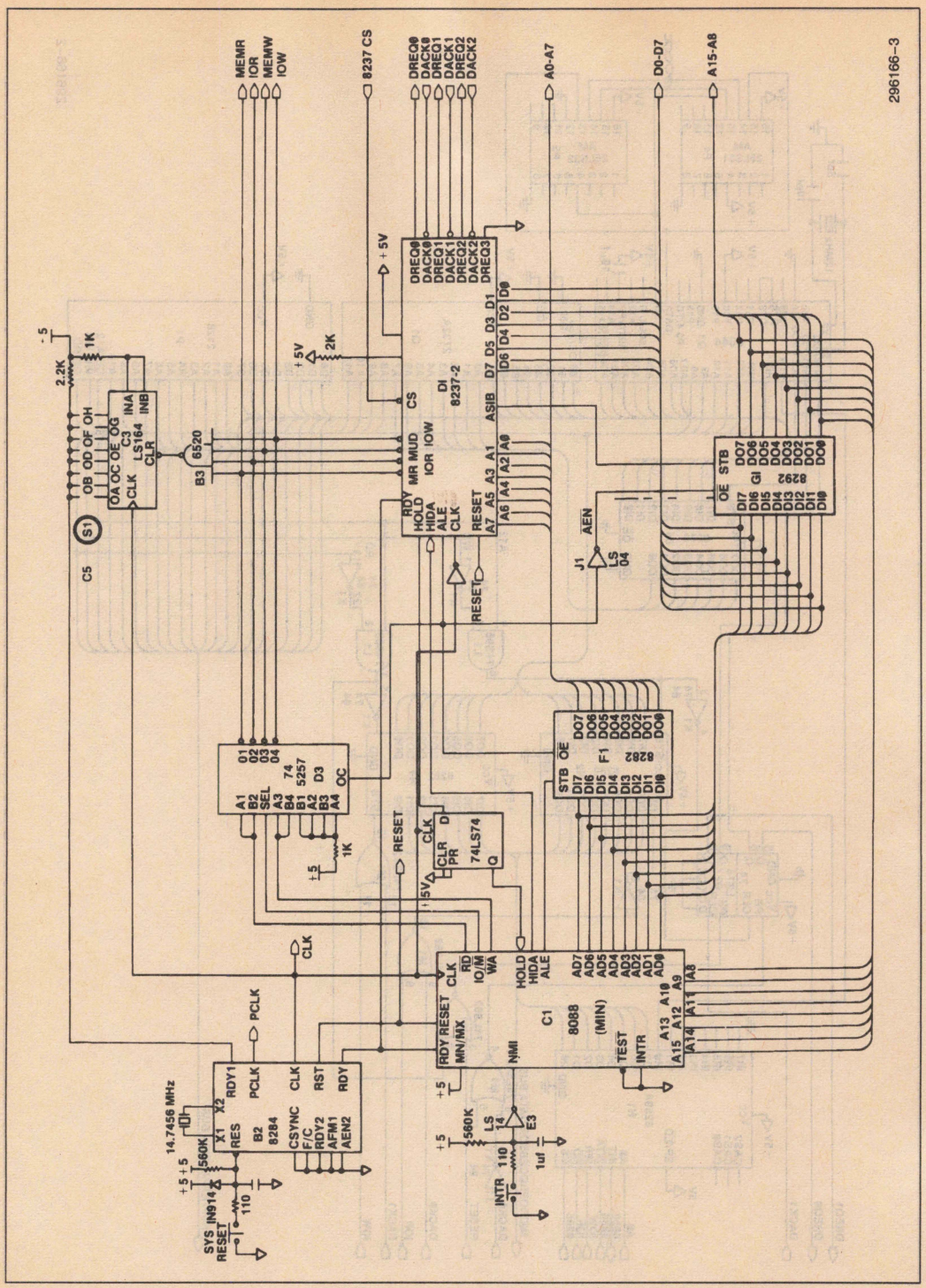


Figure 3. 8088 Min Mode System

Table 1. Transmit and Receive Software for an 8044/8088 System

```
LOC  OBJ       LINE     SOURCE

                   1    $debug    title    (8044/8088 INTERFACE)
                   2
                   3
  0000             4    FIRST_BYTE      BIT      0        ; FLAG
                   5
  0000             6                    ORG      0
  0000 8024        7                    SJMP     INIT
                   8
  0026             9                    ORG      26H
                  10
  0026 7581AA     11    INIT:          MOV      SP, #170          ; INITIALIZE STACK
  0029 75B800     12                   MOV      IP, #00           ; ALL INTERRUPTS ARE EQUAL PRIORITY
  002C 75C954     13                   MOV      SMD, #54H         ; TIMER 1 OVERFLOW, NRZI, PRE-FRAME SYNC
  002F 758844     14                   MOV      TCON, #44H        ; EDGE TRIGGERED EXTERNAL INTERRUPT 1
                  15                                              ; LEVEL TRIGGERED EXTERNAL INTERRUPT 0
                  16                                              ; TIMER 1 ON
  0032 758DEC     17                   MOV      TH1, #0ECH        ; INITIALIZE TIMER, 3125 BPS
  0035 758920     18                   MOV      TMOD, #20H        ; TIMER 1 AUTO RELOAD
                  19
  0038 75DC6A     20                   MOV      TBS, #106         ; SET UP SIU PARAMETER REGISTERS
  003B 75DB40     21                   MOV      TBL, #64
  003E 75CC2A     22                   MOV      RBS, #42
  0041 75CB40     23                   MOV      RBL, #64
  0044 75CE55     24                   MOV      STAD, #55H
  0047 75DA11     25                   MOV      TCB, #00010001B ; RR, P/F=1
                  26
  004A 901000     27                   MOV      DPTR, #1000H      ; DPTR POINTS TO TRI-STATE LATCH
  004D D200       28                   SETB     FIRST_BYTE        ; FLAG TO INDICATE FIRST BYTE
                  29                                              ; FOR RECEIVE INTERRUPT ROUTINE
  004F D2CE       30                   SETB     RBE               ; READY TO RECEIVE
  0051 75A894     31                   MOV      IE, #10010100B    ; ENABLE RECEIVE DMA AND SIU INTERRUPT
                  32
  0054 80FE       33                   SJMP     $                 ; WAIT HERE FOR INTERRUPTS
                  34
  0056 80FE       35    ERROR:         SJMP     ERROR
                  36 +1 $EJ
                  37    ;*************************    SUBROUTINES    *******************************
                  38
  0058 85CD29     39    SEND:          MOV      41, RFL           ; FIRST BYTE IN BLOCK IS COUNT
  005B 7929       40                   MOV      R1, #41           ; POINT TO BLOCK OF DATA
  005D AACD       41                   MOV      R2, RFL           ; LOAD COUNT
  005F 0A         42                   INC      R2
  0060 D2A8       43                   SETB     EX0               ; ENABLE DMA TRANSMIT INTERRUPT
  0062 22         44                   RET
                  45
                  46
                  47
                  48    ;*******************    INTERRUPT SERVICE ROUTINES    *********************
                  49
  0063            50    LOC_TMP        SET      $                 ; SET UP INTERRUPT TABLE JUMP
  0013            51                   ORG      0013H
  0013 020063     52                   LJMP     RECEIVE_DMA
  0063            53                   ORG      LOC_TMP
                  54
                  55    RECEIVE_DMA:
```

296166-69

Table 1. Transmit and Receive Software for an 8044/8088 System (Continued)

```
                    56
0063 10000E         57              JBC     FIRST_BYTE, L1 ; THE FIRST BYTE TRANSFERRED IS THE COUNT
                    58
0066 E0             59              MOVX    A, @DPTR       ; READ THE LATCH
0067 F6             60              MOV     @R0, A         ; PUT IT IN TRANSMIT BUFFER
0068 08             61              INC     R0
0069 DA08           62              DJNZ    R2, L2         ; AFTER READING BYTES,
                    63
006B D2CF           64              SETB    TBF            ; SEND DATA
006D D2CD           65              SETB    RTS
006F D200           66              SETB    FIRST_BYTE
0071 C2AA           67              CLR     EX1
                    68
0073 32             69      L2:     RETI
                    70
0074 786A           71      L1:     MOV     R0, #106       ; R0 IS A POINTER TO THE TRANSMIT
                    72                                     ; BUFFER STARTING ADDRESS
0076 E0             73              MOVX    A, @DPTR       ; PUT THE FIRST BYTE INTO
0077 FA             74              MOV     R2, A          ; R2 FOR THE COUNT
0078 32             75              RETI
                    76
  0079              77      LOC_TMP SET     $
0003                78              ORG     0003H
0003 020079         79              LJMP    TRANSMIT_DMA
0079                80              ORG     LOC_TMP
                    81

                    82      TRANSMIT_DMA
                    83
0079 E7             84              MOV     A, @R1         ; READ BYTE OUT OF THE RECEIVE BUFFER
007A F0             85              MOVX    @DPTR, A       ; WRITE IT TO THE LATCH
007B 09             86              INC     R1
007C DA08           87              DJNZ    R2, L3         ; WHEN ALL BYTES HAVE BEEN SENT
                    88
007E C2A8           89              CLR     IE.0           ; DISABLE INTERRUPT
0080 C294           90              CLR     P1.4           ; CAUSE 8088 INTERRUPT TO TERMINATE DMA
0082 D294           91              SETB    P1.4
0084 D2CE           92              SETB    RBE            ; ENABLE RECEIVER AGAIN
                    93
0086 32             94      L3:     RETI
                    95
                    96
                    97
  0087              98      LOC_TMP SET     $
0023                99              ORG     0023H
0023 020087        100              LJMP    SERIAL_INT
0087               101              ORG     LOC_TMP
                   102
                   103      SERIAL_INT:
                   104
0087 30CE06        105              JNB     RBE, RCV       ; WAS A FRAME RECEIVED
008A 30CF0B        106              JNB     TBF, XMIT      ; WAS A FRAME TRANSMITTED
008D 020056        107              LJMP    ERROR          ; IF NEITHER ERROR
                   108
0090 20CBC3        109      RCV:    JB      BOV, ERROR     ; IF BUFFER OVERRUN THEN ERROR
0093 1158          110              CALL    SEND           ; SEND THE FRAME TO THE 8088
0095 C2CC          111              CLR     SI
0097 32            112              RETI
                   113
0098 C2CC          114      XMIT:   CLR     SI
```

296166-70

Table 1. Transmit and Receive Software for an 8044/8088 System (Continued)

```
009A D2AA      115          SETB   EX1
009C 32        116          RETI
               117
               118          END

SYMBOL TABLE LISTING

NAME                TYPE      VALUE          ATTRIBUTES

BOV . . . . . B ADDR    00C8H.3 A
ERROR . . . . . C ADDR    0056H   A
EX0 . . . . . B ADDR    00A8H.0 A
EX1 . . . . . B ADDR    00A8H.2 A
FIRST_BYTE . . B ADDR    0020H.0 A
IE . . . . . D ADDR    00A8H   A
INIT . . . . . C ADDR    0026H   A
IP . . . . . D ADDR    00B8H   A
L1 . . . . . C ADDR    0074H   A
L2 . . . . . C ADDR    0073H   A
L3 . . . . . C ADDR    0086H   A
LOC_TMP . . . C ADDR    0087H   A
P1 . . . . . D ADDR    0090H   A
RBE . . . . . B ADDR    00C8H.6 A
RBL . . . . . D ADDR    00CBH   A
RBS . . . . . D ADDR    00CCH   A
RCV . . . . . C ADDR    0090H   A
RECEIVE_DMA . C ADDR    0063H   A
RFL . . . . . D ADDR    00CDH   A
RTS . . . . . B ADDR    00C8H.5 A
SEND . . . . . C ADDR    0058H   A
SERIAL_INT . . C ADDR    0087H   A
SI . . . . . B ADDR    00C8H.4 A
SMD . . . . . D ADDR    00C9H   A
SP . . . . . D ADDR    0081H   A
STAD . . . . . D ADDR    00CEH   A
TBF . . . . . B ADDR    00C8H.7 A
TBL . . . . . D ADDR    00DBH   A
TBS . . . . . D ADDR    00DCH   A
TCB . . . . . D ADDR    00DAH   A
TCON . . . . . D ADDR    0088H   A
TH1 . . . . . D ADDR    008DH   A
TMOD . . . . . D ADDR    0089H   A
TRANSMIT_DMA . C ADDR    0079H   A
XMIT . . . . . C ADDR    0098H   A

REGISTER BANK(S) USED: 0, TARGET MACHINE(S): 8044

ASSEMBLY COMPLETE, NO ERRORS FOUND
```

296166-71

Table 2. PL/M-86 Compiler RUPI/8088 Interface Example

```
SERIES-III PL/M-86 V1.0 COMPILATION OF MODULE RUPI_88
OBJECT MODULE PLACED IN :F1:R88.OBJ
COMPILER INVOKED BY:  PLM86.86 :F1:R88.SRC


            $DEBUG
            $TITLE  ('RUPI/8088 INTERFACE EXAMPLE')

  1         RUPI_88:DO;

  2   1         DECLARE

                LIT                 LITERALLY   'LITERALLY',
                TRUE                LIT         '01H',
                FALSE               LIT         '00H',

                RECV_BUFFER(64)     BYTE,
                XMIT_BUFFER(64)     BYTE,
                I                   BYTE,
                WAIT                BYTE,

                                /* 8237 PORTS*/

                MASTER_CLEAR_37     LIT         'OFFDDH',
                COMMAND_37          LIT         'OFFD8H',
                ALL_MASK_37         LIT         'OFFDFH',
                SINGLE_MASK_37      LIT         'OFFDAH',
                STATUS_37           LIT         'OFFD8H',
                REQUEST_REG_37      LIT         'OFFD9H',
                MODE_REG_37         LIT         'OFFDBH',
                CLEAR_BYTE_PTR_37   LIT         'OFFDCH',

                CH0_ADDR            LIT         'OFFD0H',
                CH0_COUNT           LIT         'OFFD1H',
                CH1_ADDR            LIT         'OFFD2H',
                CH1_COUNT           LIT         'OFFD3H',
                CH2_ADDR            LIT         'OFFD4H',
                CH2_COUNT           LIT         'OFFD5H',
                CH3_ADDR            LIT         'OFFD6H',
                CH3_COUNT           LIT         'OFFD7H',

                        /* 8237 BIT ASSIGNMENTS */

                CH0_SEL             LIT         '00H',
                CH1_SEL             LIT         '01H',
                CH2_SEL             LIT         '02H',
                CH3_SEL             LIT         '03H',
                WRITE_XFER          LIT         '04H',
                READ_XFER           LIT         '08H',
                DEMAND_MODE         LIT         '00H',
                SINGLE_MODE         LIT         '40H',
                BLOCK_MODE          LIT         '80H',
                SET_MASK            LIT         '04H',

            $EJECT
                                /* 8259 PORTS */

                STATUS_POLL_59      LIT         'OFFE0H',
                ICW1_59             LIT         'OFFE0H',
                OCW1_59             LIT         'OFFE1H',
                OCW2_59             LIT         'OFFE0H',
                OCW3_59             LIT         'OFFE0H',
                ICW2_59             LIT         'OFFE1H',
                ICW3_59             LIT         'OFFE1H',
                ICW4_59             LIT         'OFFE1H';

                        /* INTERRUPT SERVICE ROUTINE */

  3   1     OFF_RECV_DMA:   PROCEDURE   INTERRUPT 32;

  4   2         OUTPUT(SINGLE_MASK_37)=40H;
  5   2         WAIT=FALSE;
  6   2         END;
```

296166-4

Table 2. PL/M-86 Compiler RUPI/8088 Interface Example (Continued)

```
   7   1          DISABLE;

                        /* INITIALIZE 8237 */

   8   1          OUTPUT(MASTER_CLEAR_37)          =0;
   9   1          OUTPUT(COMMAND_37)               =040H;
  10   1          OUTPUT(ALL_MASK_37)              =0FH;
  11   1          OUTPUT(MODE_REG_37)              =(SINGLE_MODE OR WRITE_XFER OR CH0_SEL);
  12   1          OUTPUT(MODE_REG_37)              =(SINGLE_MODE OR READ_XFER OR CH1_SEL);
  13   1          OUTPUT(CLEAR_BYTE_PTR_37)        =0;
  14   1          OUTPUT(CH0_ADDR)                 =00H;
  15   1          OUTPUT(CH0_ADDR)                 =40H;
  16   1          OUTPUT(CH0_COUNT)                =64;
  17   1          OUTPUT(CH0_COUNT)                =00;
  18   1          OUTPUT(CH1_ADDR)                 =40H;
  19   1          OUTPUT(CH1_ADDR)                 =40H;
  20   1          OUTPUT(CH1_COUNT)                =64;
  21   1          OUTPUT(CH1_COUNT)                =00;

                        /* INITIALIZE 8259 */

  22   1          OUTPUT(ICW1_59)                  =13H; /*SINGLE MODE, EDGE TRIGGERED
                                                          INPUT, 8086 INTERRUPT TYPE*/
  23   1          OUTPUT(ICW2_59)                  =20H; /*INTERRUPT TYPE 32*/
  24   1          OUTPUT(ICW4_59)                  =03H; /*AUTO-EOI*/
  25   1          OUTPUT(OCW1_59)                  =0FEH; /*ENABLE INTERRUPT LEVEL 0*/

       $EJECT
  26   1          CALL SET$INTERRUPT  (32,OFF_RECV_DMA); /*LOAD INTERRUPT VECTOR LOCATION*/

  27   1          XMIT_BUFFER(0)=64; /*THE FIRST BYTE IN THE BLOCK OF DATA IS THE NUMBER
                                       OF BYTES TO BE TRANSFERED; NOT INCLUDING THE FIRST BYTE*/

  28   1          DO  I= 1 TO 64;  /* FILL UP THE XMIT_BUFFER WITH DATA */
  29   2          XMIT_BUFFER(I)=I;
  30   2          END;

  31   1          OUTPUT(ALL_MASK_37)=0FCH;    /*ENABLE CHANNEL 1 AND 2 */

  32   1          ENABLE;

  33   1          WAIT=TRUE;
  34   1          DO WHILE WAIT;
  35   2          END;                    /* A BLOCK OF DATA WILL BE TRANSFERRED TO THE RUPI.
                                             WHEN THE RUPI RECEIVES A BLOCK OF DATA IT WILL
                                             SEND IT TO THE 8088 MEMORY AND INTERRUPT THE 8088.
                                             THE INTERRUPT SERVICE ROUTINE WILL SHUT OFF THE DMA
                                             CONTROLLER AND SET 'WAIT' FALSE */

  36   1          DO WHILE 1;
  37   2          END;

  38   1      END;


MODULE INFORMATION:

     CODE AREA SIZE     = 00D7H    215D
     CONSTANT AREA SIZE = 0000H      0D
     VARIABLE AREA SIZE = 0082H    130D
     MAXIMUM STACK SIZE = 001EH     30D
     124 LINES READ
     0 PROGRAM WARNINGS
     0 PROGRAM ERRORS

END OF PL/M-86 COMPILATION
```

296166-5

A HIGH PERFORMANCE NETWORK USING THE 8044

2.0 INTRODUCTION

This section describes the design of an SDLC data link using the 8044 (RUPI) to implement a primary station and a secondary station. The design was implemented and tested. The following discussion assumes that the reader understands the 8044 and SDLC. This section is divided into two parts. First the data link design example is discussed. Second the software modules used to implement the data link are described. To help the reader understand the discussion of the software, flow charts and software listings are displayed in Appendix A and Appendix B, respectively.

APPLICATION DESCRIPTION

This particular data link design example uses a two wire half-duplex multidrop topology as shown in Figure 4. In an SDLC multidrop topology the primary station communicates with each secondary station. The secondary stations communicate only to the primary. Because of this hierarchial architecture, the logical topology for an SDLC multidrop is a star as shown in Figure 5. Although the physical topology of this data link is multidrop, the easiest way to understand the information flow is to think of the logical (star) topology. The term data link in this case refers to the logical communication pathways between the primary station and the secondary stations. The data links are shown in Figure 5 as two way arrows.

The application example uses dumb async terminals to interface to the SDLC network. Each secondary station has an async terminal connected to it. The secondary stations are in effect protocol converters which allows any async terminal to communicate with any other async terminal on the network. The secondary stations use an 8044 with a UART to convert SDLC to async. Figure 6 displays a block diagram of the data link. The primary station, controls the data link. In addition to data link control the primary provides a higher level layer which is a path control function or networking layer. The primary serves as a message exchange or switch. It receives information from one secondary station and retransmits it to another secondary station. Thus a virtual end to end connection is made between any two secondary stations on the network.

Three separate software modules were written for this network. The first module is a Secondary Station Driver (SSD) which provides an SDLC data link interface and a user interface. This module is a general purpose driver which requires application software to run it. The user interface to the driver provides four functions: OPEN, CLOSE, TRANSMIT, and SIU_RECV. Using these four functions properly will allow any application software to communicate over this SDLC data link without knowing the details of SDLC. The secondary station driver uses the 8044's AUTO mode.

The second module is an example of application software which is linked to the secondary station driver. This module drives the 8215A, buffers data, and interfaces with the secondary station driver's user interface.

The third module is a primary station, which is a standalone program (i.e., it is not linked to any other module). The primary station uses the 8044's NON-AUTO or FLEXIBLE mode. In addition to controlling the data link it acts as a message switch. Each time a secondary station transmits a frame, it places the destination address of the frame in the first byte of the information or I field. When the primary station receives a frame, it removes the first byte in the I field and retransmits the frame to the secondary station whose address matches this byte.

This network provides two complete layers of the OSI (Open Systems Interconnection) reference model: the physical layer and the data link layer. The physical layer implementation uses the RS-422 electrical interface. The mechanical medium consists of ribbon cable and connectors. The data link layer is defined by SDLC. SDLC's use of acknowledgements and frame numbering guarantees that messages will be received in the same order in which they were sent. It also guarantees message integrity over the data link. However this network will not guarantee secondary to secondary message delivery, since there are acknowledgements between secondary stations.

2.1 Hardware

The schematic of the hardware is given in Figure 7. The 8251A is used as an async communications controller, in support of the 8044. TxRDY and RxRDY on the 8251A are both tied to the two available external interrupts of the 8044 since the secondary station driver is totally interrupt driven. The 8044 buffers the data and some variables in a 2016 (2K x 8 static RAM). The 8254 programmable interval timer is employed as a programmable baud rate generator and system clock driver for the 8251A. The third output from the 8254 could be used as an external baud rate generator for the 8044. The 2732A shown in the diagram was not used

since the software for both the primary and secondary stations used far less than the 4K bytes provided on the 8744. For the async interface, the standard RS-232 mechanical and electrical interface was used. For the SDLC channel, a standard two wire three state RS-422 driver is used. A DIP switch connected to one of the available ports on the 8044 allows the baud rate, parity, and stop bits to be changed on the async interface. The primary station hardware does not use the USART, 8254, nor the RS-232 drivers.

2.2 SDLC Basic Repertoire

The SDLC commands and responses implemented in the data link include the SDLC Basic Repertoire as defined in the IBM SDLC General Information manual. Table 3 shows the commands and responses that the primary and the secondary station in this data link design recognize and send.

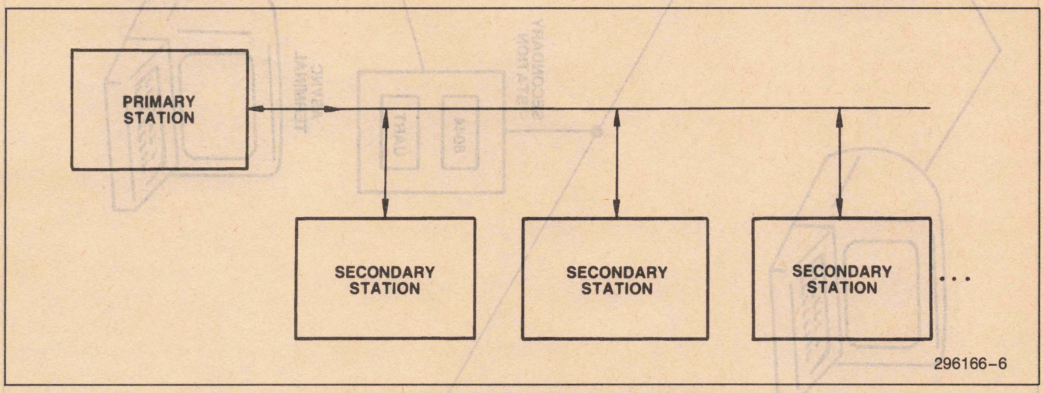


Figure 4. SDLC Multidrop Topology

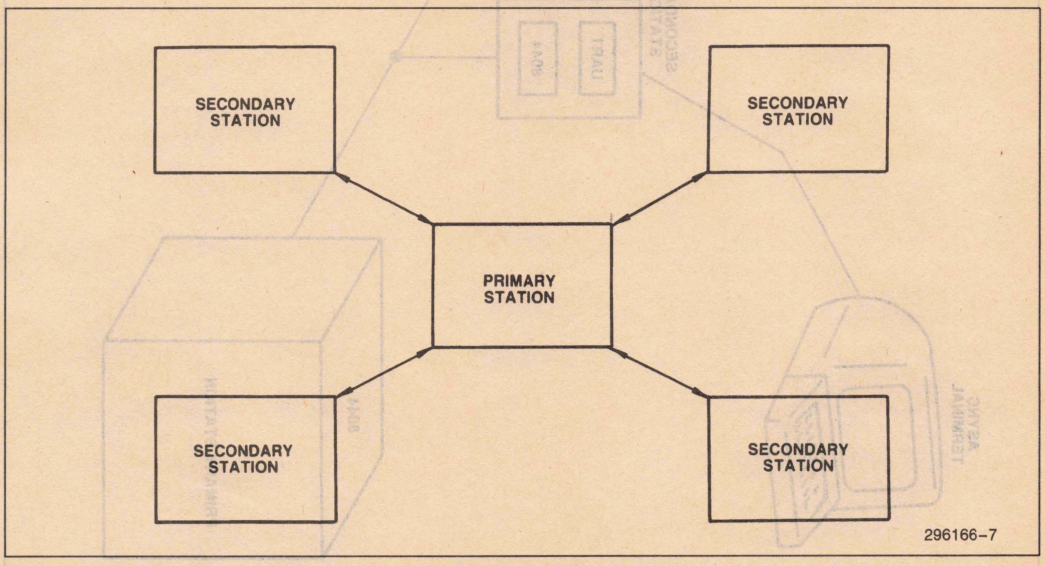


Figure 5. SDLC Logical Topology

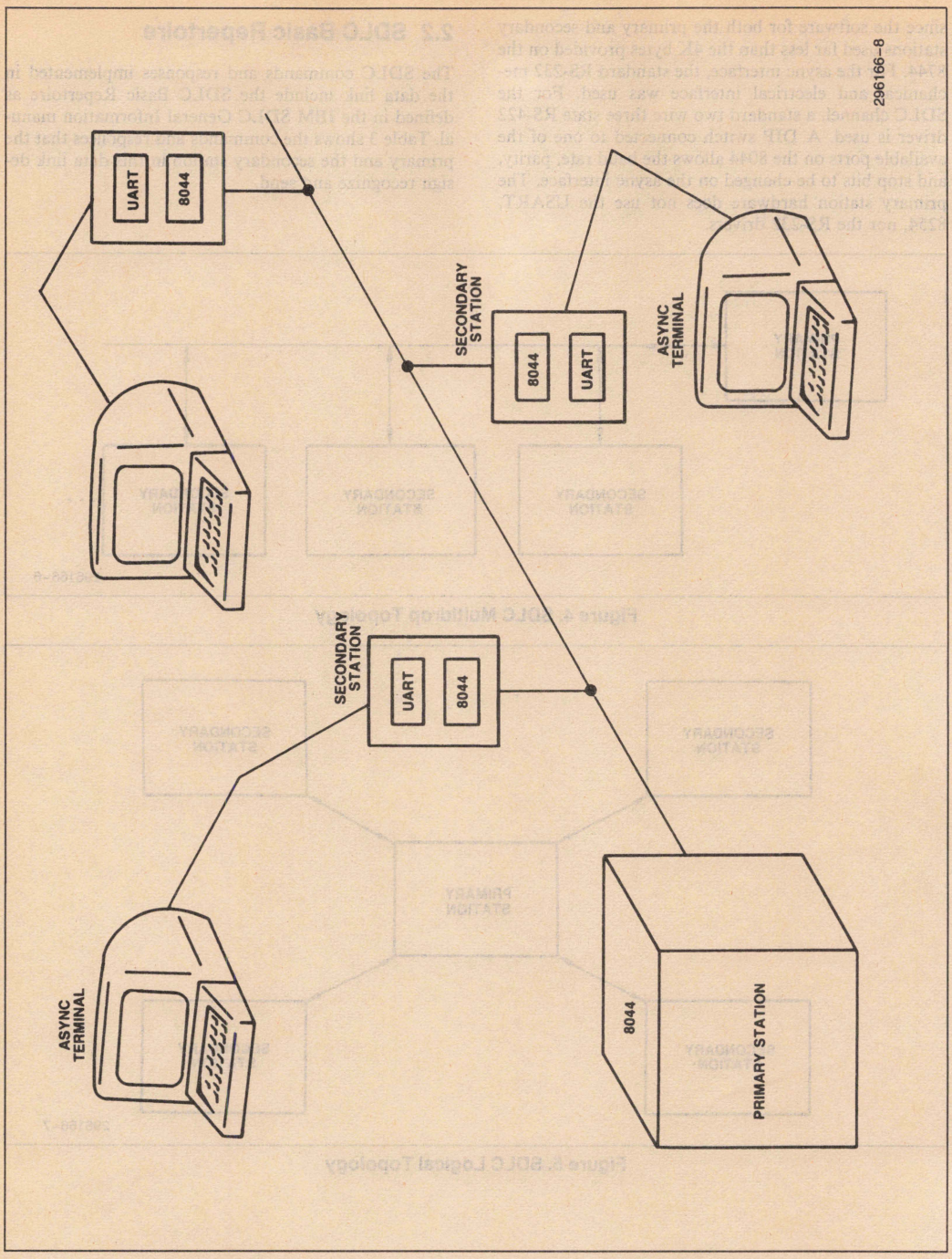


Figure 6. Block Diagram of the Data Link Application Example

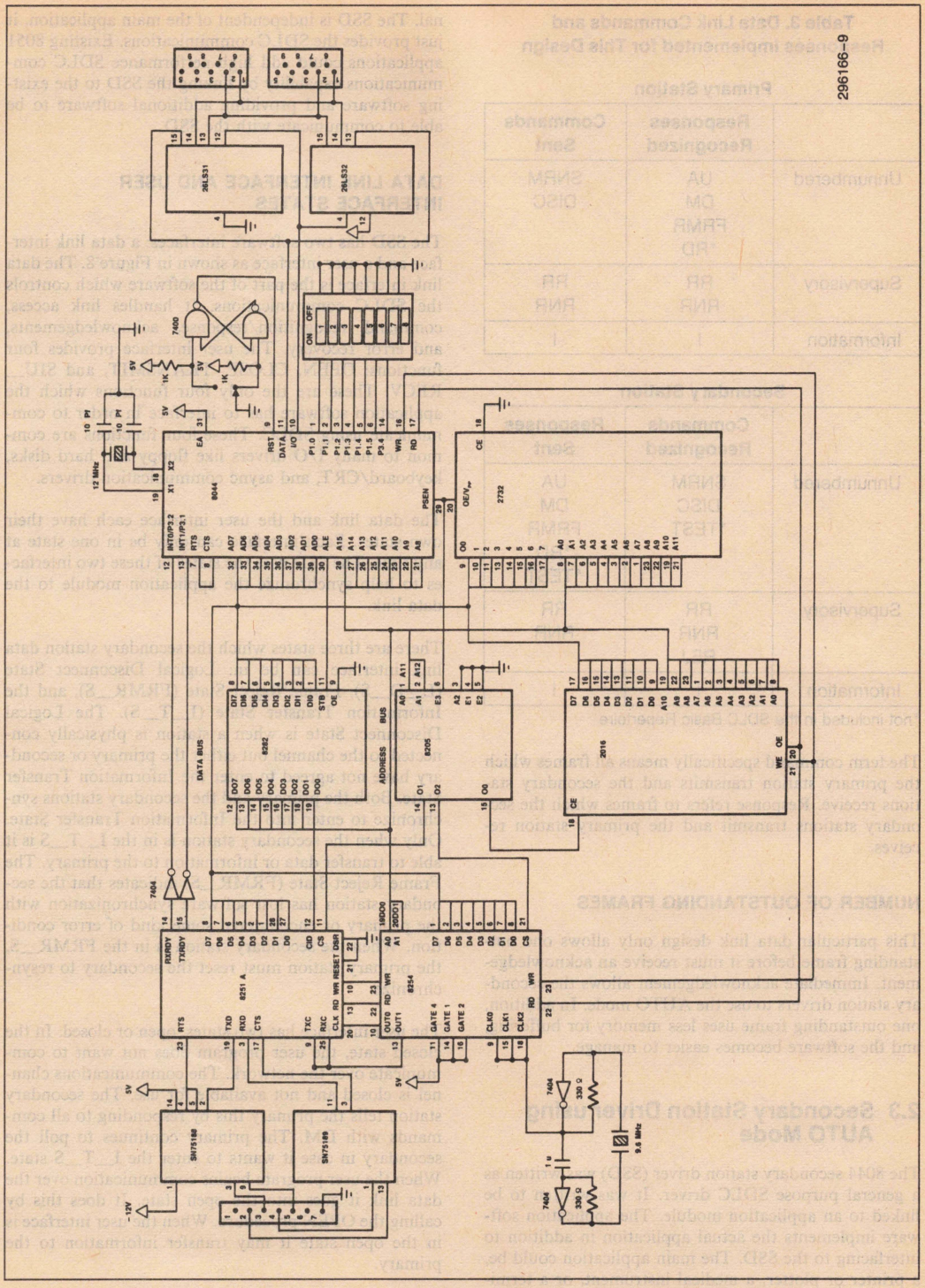


Figure 7. Schematic of Async/SDLC Secondary Station Protocol Converter

Table 3. Data Link Commands and Responses Implemented for This Design

Primary Station

	Responses Recognized	Commands Sent
Unnumbered	UA DM FRMR *RD	SNRM DISC
Supervisory	RR RNR	RR RNR
Information	I	I

Secondary Station

	Commands Recognized	Responses Sent
Unnumbered	SNRM DISC *TEST	UA DM FRMR *RD *TEST
Supervisory	RR RNR REJ	RR RNR
Information	I	I

*not included in the SDLC Basic Repertoire

The term command specifically means all frames which the primary station transmits and the secondary stations receive. Response refers to frames which the secondary stations transmit and the primary station receives.

NUMBER OF OUTSTANDING FRAMES

This particular data link design only allows one outstanding frame before it must receive an acknowledgement. Immediate acknowledgement allows the secondary station drivers to use the AUTO mode. In addition, one outstanding frame uses less memory for buffering, and the software becomes easier to manage.

2.3 Secondary Station Driver using AUTO Mode

The 8044 secondary station driver (SSD) was written as a general purpose SDLC driver. It was written to be linked to an application module. The application software implements the actual application in addition to interfacing to the SSD. The main application could be, a printer or plotter, a medical instrument, or a terminal. The SSD is independent of the main application, it just provides the SDLC communications. Existing 8051 applications could add high performance SDLC communications capability by linking the SSD to the existing software and providing additional software to be able to communicate with the SSD.

DATA LINK INTERFACE AND USER INTERFACE STATES

The SSD has two software interfaces: a data link interface and a user interface as shown in Figure 8. The data link interface is the part of the software which controls the SDLC communications. It handles link access, command recognition/response, acknowledgements, and error recovery. The user interface provides four functions: OPEN, CLOSE, TRANSMIT, and SIU_RECV. These are the only four functions which the application software has to interface in order to communicate using SDLC. These four functions are common to many I/O drivers like floppy and hard disks, keyboard/CRT, and async communication drivers.

The data link and the user interface each have their own states. Each interface can only be in one state at any time. The SSD uses the states of these two interfaces to help synchronize the application module to the data link.

There are three states which the secondary station data link interface can be in: Logical Disconnect State (L_D_S), Frame Reject State (FRMR_S), and the Information Transfer State (I_T_S). The Logical Disconnect State is when a station is physically connected to the channel but either the primary or secondary have not agreed to enter the Information Transfer State. Both the primary and the secondary stations synchronize to enter into the Information Transfer State. Only when the secondary station is in the I_T_S is it able to transfer data or information to the primary. The Frame Reject State (FRMR_S) indicates that the secondary station has lost software synchronization with the primary or encountered some kind of error condition. When the secondary station is in the FRMR_S, the primary station must reset the secondary to resynchronize.

The user interface has two states, open or closed. In the closed state, the user program does not want to communicate over the network. The communications channel is closed and not available for use. The secondary station tells the primary this by responding to all commands with DM. The primary continues to poll the secondary in case it wants to enter the I_T_S state. When the user program begins communication over the data link it goes into the open state. It does this by calling the OPEN procedure. When the user interface is in the open state it may transfer information to the primary.

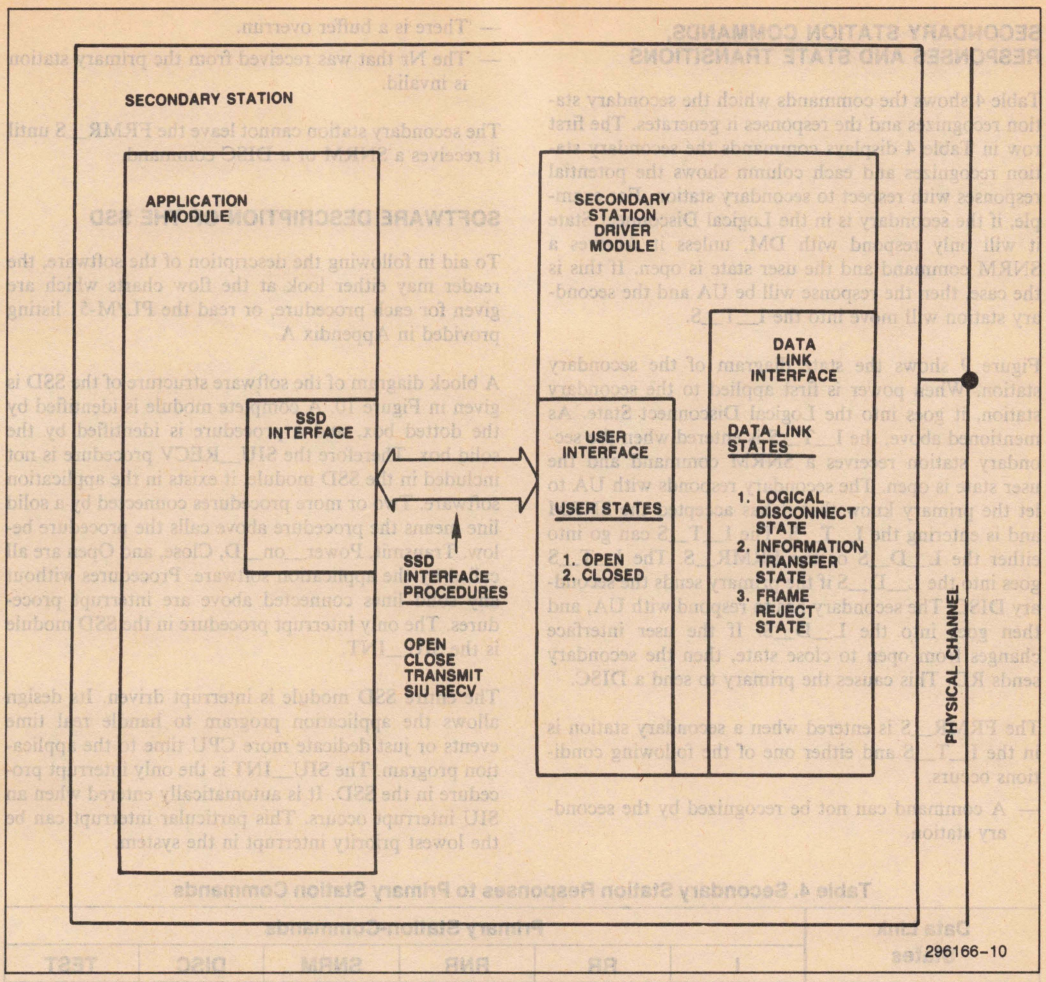


Figure 8. Secondary Station Software Modules

SECONDARY STATION COMMANDS, RESPONSES AND STATE TRANSITIONS

Table 4 shows the commands which the secondary station recognizes and the responses it generates. The first row in Table 4 displays commands the secondary station recognizes and each column shows the potential responses with respect to secondary station. For example, if the secondary is in the Logical Disconnect State it will only respond with DM, unless it receives a SNRM command and the user state is open. If this is the case, then the response will be UA and the secondary station will move into the I_T_S.

Figure 9 shows the state diagram of the secondary station. When power is first applied to the secondary station, it goes into the Logical Disconnect State. As mentioned above, the I_T_S is entered when the secondary station receives a SNRM command and the user state is open. The secondary responds with UA to let the primary know that it has accepted the SNRM and is entering the I_T_S. The I_T_S can go into either the L_D_S or the FRMR_S. The I_T_S goes into the L_D_S if the primary sends the secondary DISC. The secondary has to respond with UA, and then goes into the L_D_S. If the user interface changes from open to close state, then the secondary sends RD. This causes the primary to send a DISC.

The FRMR_S is entered when a secondary station is in the I_T_S and either one of the following conditions occurs.

— A command can not be recognized by the secondary station.

— There is a buffer overrun.

— The Nr that was received from the primary station is invalid.

The secondary station cannot leave the FRMR_S until it receives a SNRM or a DISC command.

SOFTWARE DESCRIPTION OF THE SSD

To aid in following the description of the software, the reader may either look at the flow charts which are given for each procedure, or read the PL/M-51 listing provided in Appendix A.

A block diagram of the software structure of the SSD is given in Figure 10. A complete module is identified by the dotted box, and a procedure is identified by the solid box. Therefore the SIU_RECV procedure is not included in the SSD module, it exists in the application software. Two or more procedures connected by a solid line means the procedure above calls the procedure below. Transmit, Power_on_D, Close, and Open are all called by the application software. Procedures without any solid lines connected above are interrupt procedures. The only interrupt procedure in the SSD module is the SIU_INT.

The entire SSD module is interrupt driven. Its design allows the application program to handle real time events or just dedicate more CPU time to the application program. The SIU_INT is the only interrupt procedure in the SSD. It is automatically entered when an SIU interrupt occurs. This particular interrupt can be the lowest priority interrupt in the system.

Table 4. Secondary Station Responses to Primary Station Commands

Data Link States	Primary Station-Commands					
	I	RR	RNR	SNRM	DISC	TEST
Information Transfer State	I RR RNR RD FRMR	I RR RNR RD FRMR	I RR RNR RD FRMR	RD UA	UA	RD Test
Logical Disconnect State	DM	DM	DM	DM UA	DM	DM
Frame Reject State	FRMR	FRMR	FRMR	UA	UA	FRMR

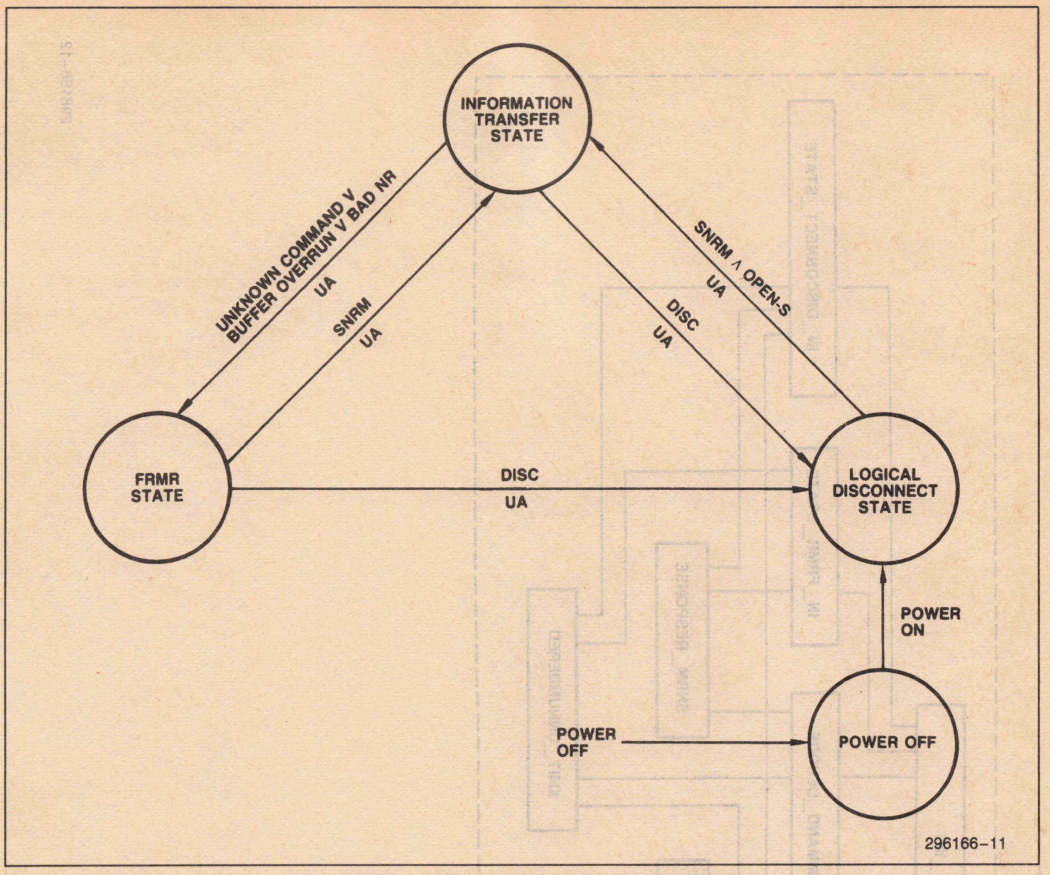


Figure 9. State Diagram of Secondary Station

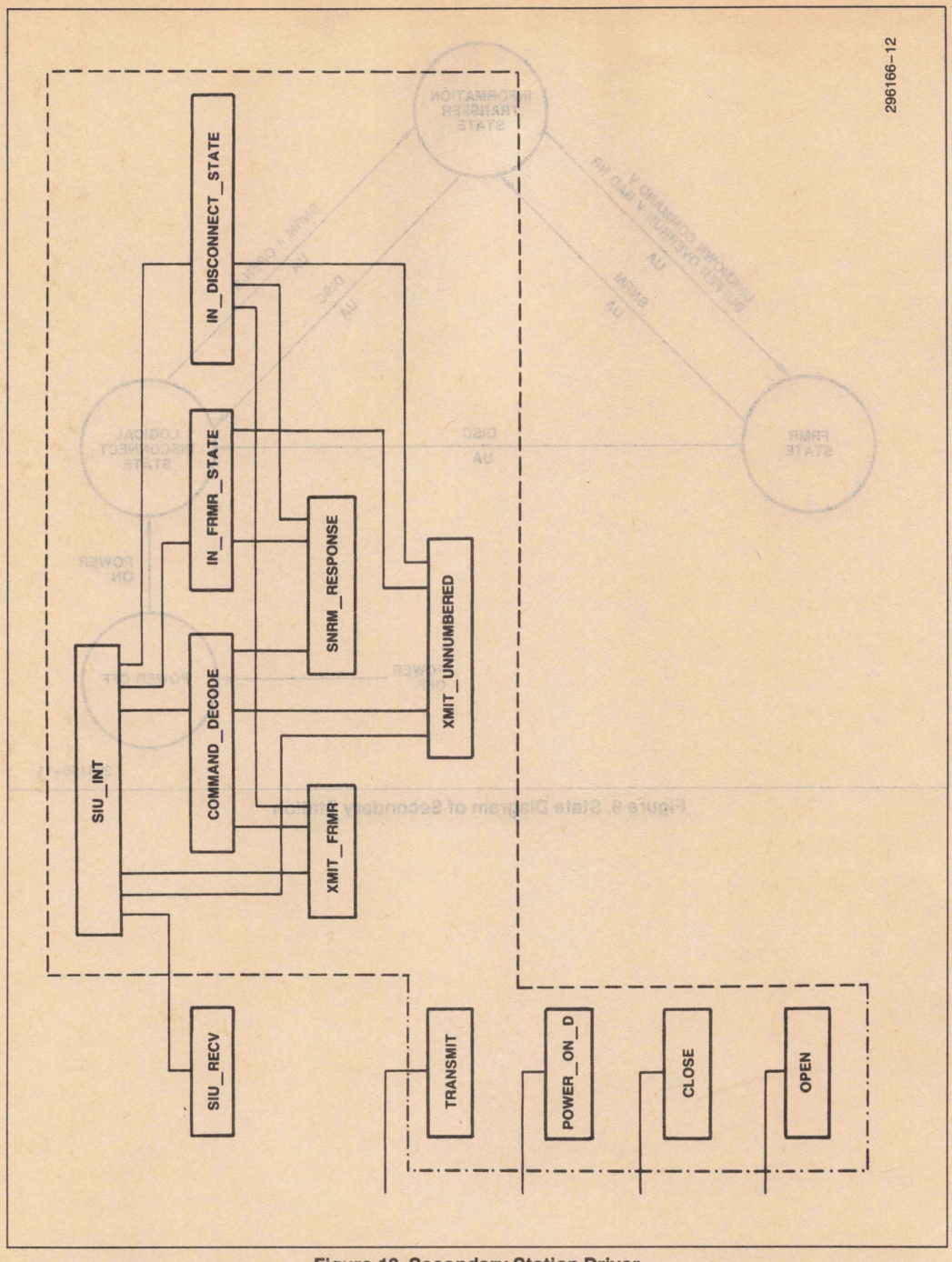


Figure 10. Secondary Station Driver

SSD INITIALIZATION

Upon reset the application software is entered first. The application software initializes its own variables then calls Power_On_D which is the SSD's initialization routine. The SSD's initialization sets up the transmit and receive data buffer pointers (TBS and RBS), the receive buffer length (RBL), and loads the State variables. The STATION_STATE begins in the L_D_S state, and the USER_STATE begins in the closed state. Finally Power_On_D initializes XMIT_BUFFER_EMPTY which is a bit flag. This flag serves as a semaphore between the SSD and the application software to indicate the status of the on chip transmit buffer. The SSD does not set the station address. It is the application software's responsibility to do this. After initialization, the SSD is read to respond to all of the primary station commands. Each time a frame is received with a matching station address and a good CRC, the SIU_INT procedure is entered.

SIU_INT PROCEDURE

The first thing the SIU_INT procedure clears is the serial interrupt_bit (SI) in the STS register. If the SIU_INT procedure returns with this bit set, another SI interrupt will occur.

The SIU_INT procedure is branches three independent cases. The first case is entered if the STATION_STATE is not in the I_T_S. If this is true, then the SIU is not in the AUTO mode, and the CPU will have to respond to the primary on its own. (Remember that the AUTO mode is entered when the STATION_STATE enters into I_T_S.) If the STATION_STATE is in the I_T_S, then either the SIU has just left the AUTO mode, or is still in the AUTO mode. This is the second and third case, respectively.

In the first case, if the STATION_STATE is not in the I_T_S, then it must be in either the L_D_S or the FRMR_S. In either case a separate procedure is called based on which state the station is in. The In_Disconnect_State procedure sends to the primary a DM response, unless it received a SNRM command and the USER_STATE equals open. In that case the SIU sends a UA and enters into the I_T_S. The In_FRMR_State procedure will send the primary the FRMR response unless it received either a DISC or an SNRM. If the primary's command was a DISC, then the secondary will send a UA and enter into the L_D_S. If the primary's command was a SNRM, then the secondary will send a UA, enter into the I_T_S, and clear NSNR register.

For the second case, if the STATION_STATE is in the I_T_S but the SIU left the AUTO mode, then the CPU must determine why the AUTO mode was exited, and generate a response to the primary. There are four reasons for the SIU to automatically leave the AUTO mode. The following is a list of these reasons, and the responses given by the SSD based on each reason.

1. The SIU has received a command field it does not recognize.

   Response: If the CPU recognizes the command, it generates the appropriate response. If neither the SIU nor the CPU recognize the command, then a FRMR response is sent.

2. The SIU has received a Sequence Error Sent (SES = 1 in NSNR register). Nr(P) ≠ Ns(S) + 1, and Nr(P) ≠ Ns(S).

   Response: Send FRMR.

3. A buffer overrun has occurred. BOV = 1 in STS register.

   Response: Send FRMR.

4. An I frame with data was received while RPB = 1.

   Response: Go back into AUTO mode and send an AUTO mode response

In addition to the above reasons, there is one condition where the CPU forces the SIU out of the AUTO mode. This is discussed in the SSD's User Interface Procedures section in the CLOSED procedure description

Finally, case three is when the STATION_STATE is in the I_T_S and the AUTO mode. The CPU first looks at the TBF bit. If this bit is 0 then the interrupt may have been caused by a frame which was transmitted and acknowledged. Therefore the XMIT_BUFFER_EMPTY flag is set again, indicating that the application software can transmit another frame.

The other reason this section of code could be entered is if a valid I frame was received. When a good I frame is received the RBE bit equals 0. This means that the receiver is disabled. If the primary were to poll the 8044 while RBE = 0, it would time out since no response would be given. Time outs reduce network throughput. To improve network performance, the CPU first sets RBP, then sets RBE. Now when the primary polls the 8044 an immediate RNR response is given. At this point the SSD calls the application software procedure SIU_RECV and passes the length of the data as a parameter. The SIU_RECV procedure reads the data out of the receive buffer then returns to the SSD module. Now that the receive information has been transferred, RBP can be cleared.

COMMAND_DECODE PROCEDURE

The Command_Decode procedure is called from the SIU_INT procedure when the STATION_STATE = I_T_S and the SIU left the AUTO mode as a result of not being able to recognize the receive control byte. Commands which the SIU AUTO mode does not

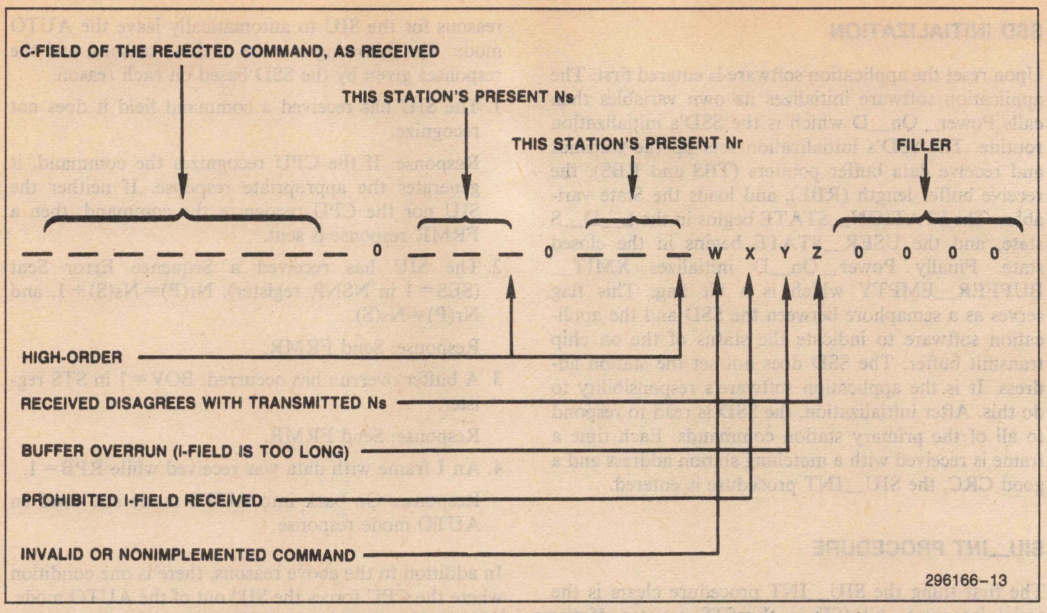


Figure 11. Information Field of the FRMR Response, as Transmitted

recognize are handled here. The commands recognized in this procedure are: SNRM, DISC, and TEST. Any other command received will generate a Frame Reject with the nonimplemented command bit set in the third data byte of the FRMR frame. Any additional unnumbered frame commands which the secondary station is going to implement, should be implemented in this procedure.

IF an SNRM is received the command_decode procedure calls the SNRM_Response procedure. The SNRM_Response procedure sets the STATION_STATE = I_T_S, clears the NSNR register and responds with a UA frame. If a DISC is received, the command_decode procedure sets the STATION_STATE = L_D_S, and responds with a UA frame. When a TEST frame is received, and there is no buffer overrun, the command_decode procedure responds with a TEST frame retransmitting the same data it received. However if a TEST frame is received and there is a buffer overrun, then a TEST frame will be sent without any data, instead of a FRMR with the buffer overrun bit set.

FRAME REJECT PROCEDURES

There are two procedures which handle the FRMR state: XMIT_FRMR and IN_FRMR_STATE. XMIT_FRMR is entered when the secondary station first goes into the FRMR state. The frame reject response frame contains the FRMR response in the command field plus three additional data bytes in the I field. Figure 11 displays the format for the three data bytes in the I field of a FRMR response. The XMIT_FRMR procedure sets up the Frame Reject response frame based on the parameter REASON which is passed to it. Each place in the SSD code that calls the XMIT_FRMR procedure, passes the REASON that this procedure was called, which in turn is communicated to the primary station. The XMIT_FRMR procedure uses three bytes of internal RAM which it initializes for the correct response. The TBS and TBL registers are then changed to point to the FRMR buffer so that when a response is sent these three bytes will be included in the I field.

The IN_FRMR_STATE procedure is called by the SIU_INT procedure when the STATION_STATE already is in the FRMR state and a response is required. The IN_FRMR_STATE procedure will only allow two commands to remove the secondary station from the FRMR state: SNRM and DISC. Any other command which is received while in the FRMR state will result in a FRMR response frame.

XMIT_UNNUMBERED PROCEDURE

This is a general purpose transmit procedure, used only in the FLEXIBLE mode, which sends unnumbered responses to the primary. It accepts the control byte as a parameter, and also expects the TBL register to be set before the procedure is called. This procedure waits until the frame has been transmitted before returning. If

this procedure returned before the transmit interrupt was generated, the SIU_INT routine would be entered. The SIU_INT routine would not be able to distinguish this condition.

SSD's User Interface Procedures—OPEN, CLOSE, TRANSMIT, SIU_RECV—are discussed in the following section.

The OPEN procedure is the simplest of all, it changes the USER_STATE to OPEN_S then returns. This lets the SSD know that the user wants to open the channel for communications. When the SSD receives a SNRM command, it checks the USER_STATE. If the USER_STATE is open, then the SSD will respond with a UA, and the STATION_STATE enters the I_T_S.

The CLOSE procedure is also simple, it changes the USER_STATE to CLOSED_S and sets the AM bit to 0. Note that when the CPU sets the AM bit to 0 it puts the SIU out of the AUTO mode. This event is asynchronous to the events on the network. As a result an I frame can be lost. This is what can happen.

1. AM is set to 0 by the CLOSE Procedure.
2. An I frame is received and an SI interrupt occurs.
3. The SIU_INT procedure enters case 2 (STATION_STATE = I_T_S, and AM = 0).
4. Case 2 detects that the USER_STATE = CLOSED_S, sends an RD response and ignores the fact that an I frame was received.

Therefore it is advised to never call the CLOSE procedure or take the SIU out of the AUTO mode when it is receiving I frames or an I frame will be lost.

For both the TRANSMIT and SIU_RECV procedures, it is the application software's job to put data into the transmit buffer, and take data out of the receive buffer. The SSD does not transfer data in or out of its transmit or receive buffers because it does not know what kind of buffering the application software is implementing. What the SSD does do is notify the application software when the transmit buffer is empty, XMIT_BUFFER_EMPTY = 1, and when the receive buffer is full.

One of the functions that the SSD performs to synchronize the application software to the SDLC data link. However some of the synchronization must also be done by the application software. Remember that the SSD does not want to hang up the application software waiting for some event to occur on the SDLC data link, therefore the SSD always returns to the application software as soon as possible.

For example, when the application software calls the OPEN procedure, the SSD returns immediately. The application software thinks that the SDLC channel is now open and it can transmit. This is not the case. For the channel to be open, the SSD must receive an SNRM from the primary and respond with a UA. However, the SSD does not want to hang up the application software waiting for an SNRM from the primary before returning from the OPEN procedure. When the TRANSMIT procedure is called, the SSD expects the STATION_STATE to be in the I_T_S. If it isn't, the SSD refuses to transmit the data. The TRANSMIT procedure first checks to see if the USER_STATE is open. If not, the USER_STATE_CLOSED parameter is passed back to the application module. The next thing TRANSMIT checks is the STATION_STATE. If this is not open, then TRANSMIT passes back LINK_DISCONNECTED. This means that the USER_STATE is open, but the SSD hasn't received an SNRM command from the primary yet. Therefore, the application software should wait awhile and try again. Based on network performance, one knows the maximum amount of time it will take for a station to be polled. If the application software waits this length of time and tries again but still gets a LINK_DISCONNECTED parameter passed back, higher level recovery must be implemented.

Before loading the transmit buffer and calling the TRANSMIT procedure, the application software must check to see that XMIT_BUFFER_EMPTY = 1. This flag tells the application software that it can write new data into the transmit buffer and call the TRANSMIT procedure. After the application software has verified that XMIT_BUFFER_EMPTY = 1, it fills the transmit buffer with the data and calls the TRANSMIT procedure passing the length of the buffer as a parameter. The TRANSMIT procedure checks for three reasons why it might not be able to transmit the frame. If any of these three reasons are true, the TRANSMIT procedure returns a parameter explaining why it couldn't send the frame. If the application software receives one of these responses, it must rectify the problem and try again. Assuming these three conditions are false, then the SSD clears XMIT_BUFFER_EMPTY, attempts to send the data and returns the parameter DATA_TRANSMITTED. XMIT_BUFFER_EMPTY will not be set to 1 again until the data has been transmitted and acknowledged.

The SIU_RECV procedure must be incorporated into the application software module. When a valid I frame is received by the SIU, it calls the SIU_RECV procedure and passes the length of the received data as a parameter. The SIU_RECV procedure must remove all of the data from the receive buffer before returning to the SIU_INT procedure.

LINKING UP TO THE SSD

Figure 12 shows the necessary parts to include in a PL/M-51 application program that will be linked to the SSD module. RL51 is used to link and locate the SSD and application modules. The command line used to do this is:

```
RL51 SSD.obj,filename.obj,PLM51.LIB TO
filename & RAMSIZE(192)

$registerbank(0)
user$mod: do;
$include (reg44.dcl)
declare
 lit                literally 'literally',
 buffer_length      lit         '60',
 siu_xmit_buffer
 (buffer_length)    byte   external idata,
 siu_recv_buffer
 (buffer_length)    byte        external,
 xmit_buffer_empty bit          external;

/* external procedures */

power_on_d: procedure  external;
end power_on_d;

close: procedure       external using 1;
end close;

open: procedure        external using 1;
end open;

transmit: procedure
(xmit_buffer_length) byte      external;
 declare xmit_buffer_length     byte;
end transmit;

/* local procedures */

siu_recv: procedure (length)    using 1;
public
 declare   length byte,
        •
        •
        •
end siu_recv;
```

Figure 12. Applications Module Link Information

PL/M-51 AND REGISTER BANKS

The 8044 has four register banks. PL/M-51 assumes that an interrupt procedure never uses the same bank as the procedure it interrupts. The USING attribute of a procedure, or the $REGISTERBANK control, can be used to ensure that.

The SSD module uses the $REGISTERBANK(1) attribute. Some procedures are modified with the USING attribute based on the register bank level of the calling procedure.

2.4 Application Module; ASYNC to SDLC Protocol Converter

One of the purposes of this application module is to demonstrate how to interface software to the SSD. Another purpose is to implement and test a practical application. This application software performs I/O with an async terminal through a USART, buffers data, and also performs I/O with the SSD. In addition, it allows the user on the async terminal to: set the station address, set the destination address, and go online and offline. Setting the station address sets the byte in the STAD register. The destination address is the first byte in the I field. Going online or offline results in either calling the OPEN or CLOSE procedure respectively.

After the secondary station powers up, it enters the 'terminal mode', which accepts data from the terminal. However, before any data is sent, the user must configure the station. The station address and destination address must be set, and the station must be placed online. To configure the station the ESC character is entered at the terminal which puts the protocol converter into the 'configure mode'. Figure 13 shows the menu which appears on the terminal screen.

```
(/)8044 Secondary Station
 /

1 - Set the Station Address
2 - Set the Destination Address
3 - Go Online
4 - Go Offline
5 - Return to terminal mode
    Enter option __
```

Figure 13. Menu for the Protocol Converter

In the terminal mode data is buffered up in the secondary station. A Line Feed character 'LF' tells the secondary station to send an I frame. If more than 60 bytes are buffered in the secondary station when a 'LF' is received, the applications software packetizes the data into 60 bytes or less per frame. If a LF is entered when the station is offline, an error message comes on the screen which says 'Unable to Get Online'.

The secondary station also does error checking on the async interface for Parity, Framing Error, and Overrun Error. If one of these errors are detected, an error message is displayed on the terminal screen.

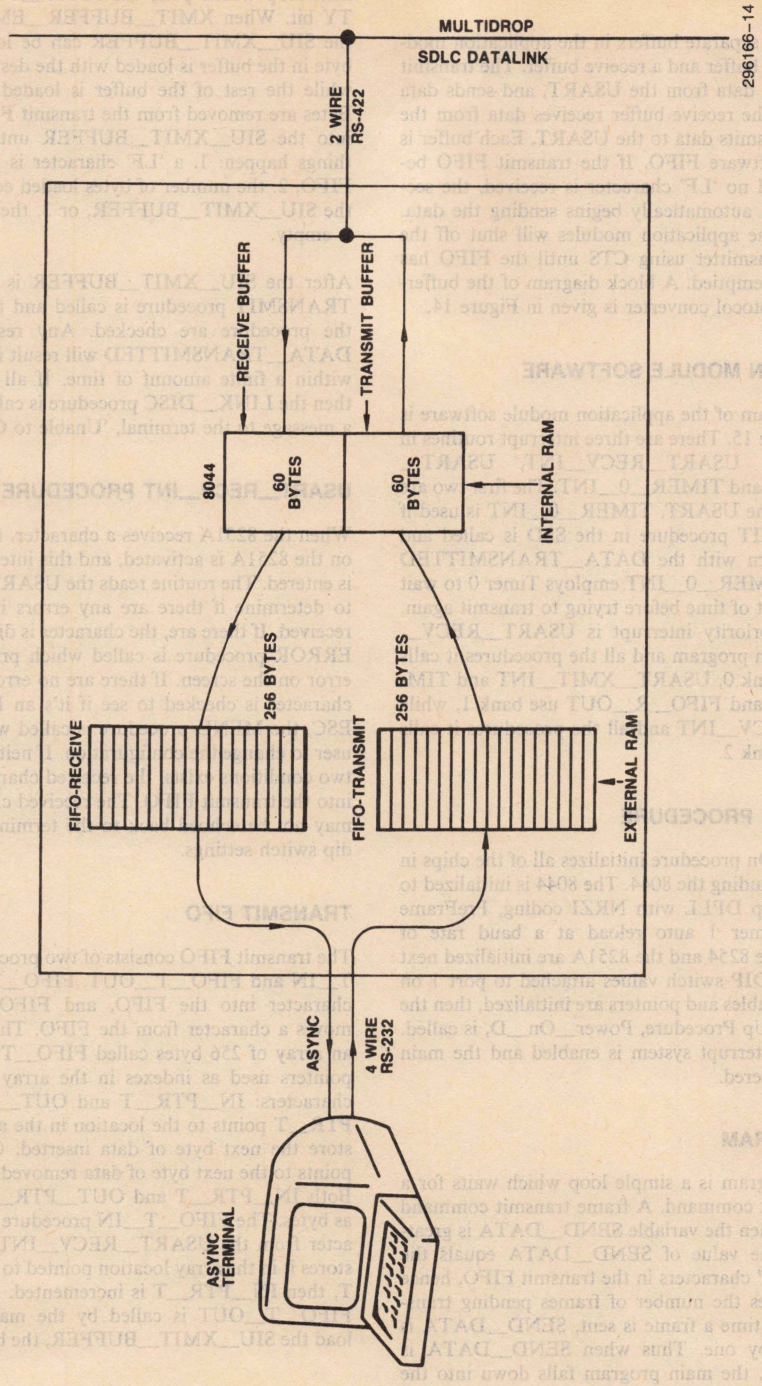


Figure 14. Block Diagram of Secondary Station Protocol Converter Illustrating Buffering

BUFFERING

There are two separate buffers in the application module: a transmit buffer and a receive buffer. The transmit buffer receives data from the USART, and sends data to the SSD. The receive buffer receives data from the SSD, and transmits data to the USART. Each buffer is a 256 byte software FIFO. If the transmit FIFO becomes full and no 'LF' character is received, the secondary station automatically begins sending the data. In addition, the application modules will shut off the terminal's transmitter using CTS until the FIFO has been partially emptied. A block diagram of the buffering for the protocol converter is given in Figure 14.

APPLICATION MODULE SOFTWARE

A block diagram of the application module software is given in Figure 15. There are three interrupt routines in this module: USART_RECV_INT, USART_XMIT_INT, and TIMER_0_INT. The first two are for servicing the USART. TIMER_0_INT is used if the TRANSMIT procedure in the SSD is called and does not return with the DATA_TRANSMITTED parameter. TIMER_0_INT employs Timer 0 to wait a finite amount of time before trying to transmit again. The highest priority interrupt is USART_RECV_INT. The main program and all the procedures it calls use register bank 0, USART_XMIT_INT and TIMER_0_INT and FIFO_R_OUT use bank 1, while USART_RECV_INT and all the procedures it calls use register bank 2.

POWER_ON PROCEDURE

The Power_On procedure initializes all of the chips in the system including the 8044. The 8044 is initialized to use the on-chip DPLL with NRZI coding, PreFrame Sync, and Timer 1 auto reload at a baud rate of 62.5 Kbps. The 8254 and the 8251A are initialized next based on the DIP switch values attached to port 1 on the 8044. Variables and pointers are initialized, then the SSD's Power-Up Procedure, Power_On_D, is called. Finally, the interrupt system is enabled and the main program is entered.

MAIN PROGRAM

The main program is a simple loop which waits for a frame transmit command. A frame transmit command is indicated when the variable SEND_DATA is greater than 0. The value of SEND_DATA equals the number of 'LF' characters in the transmit FIFO, hence it also indicates the number of frames pending transmission. Each time a frame is sent, SEND_DATA is decremented by one. Thus when SEND_DATA is greater than 0, the main program falls down into the next loop which polls the XMIT_BUFFER_EMPTY bit. When XMIT_BUFFER_EMPTY equals 1, the SIU_XMIT_BUFFER can be loaded. The first byte in the buffer is loaded with the destination address while the rest of the buffer is loaded with the data. Bytes are removed from the transmit FIFO and placed into the SIU_XMIT_BUFFER until one of three things happen: 1. a 'LF' character is read out of the FIFO, 2. the number of bytes loaded equals the size of the SIU_XMIT_BUFFER, or 3. the transmit FIFO is empty.

After the SIU_XMIT_BUFFER is filled, the SSD TRANSMIT procedure is called and the results from the procedure are checked. Any result other than DATA_TRANSMITTED will result in several retries within a finite amount of time. If all the retries fail, then the LINK_DISC procedure is called which sends a message to the terminal, 'Unable to Get Online'.

USART_RECV_INT PROCEDURE

When the 8251A receives a character, the RxRDY pin on the 8251A is activated, and this interrupt procedure is entered. The routine reads the USART status register to determine if there are any errors in the character received. If there are, the character is discarded and the ERROR procedure is called which prints the type of error on the screen. If there are no errors, the received character is checked to see if it's an ESC. If it is an ESC, the MENU procedure is called which allows the user to change the configuration. If neither one of these two conditions exists, the received character is inserted into the transmit FIFO. The received character may or may not be echoed back to the terminal based on the dip switch settings.

TRANSMIT FIFO

The transmit FIFO consists of two procedures: FIFO_T_IN and FIFO_T_OUT. FIFO_T_IN inserts a character into the FIFO, and FIFO_T_OUT removes a character from the FIFO. The FIFO itself is an array of 256 bytes called FIFO_T. There are two pointers used as indexes in the array to address the characters: IN_PTR_T and OUT_PTR_T. IN_PTR_T points to the location in the array which will store the next byte of data inserted. OUT_PTR_T points to the next byte of data removed from the array. Both IN_PTR_T and OUT_PTR_T are declared as bytes. The FIFO_T_IN procedure receives a character from the USART_RECV_INT procedure and stores it in the array location pointed to by IN_PTR_T, then IN_PTR_T is incremented. Similarly, when FIFO_T_OUT is called by the main program, to load the SIU_XMIT_BUFFER, the byte in the array

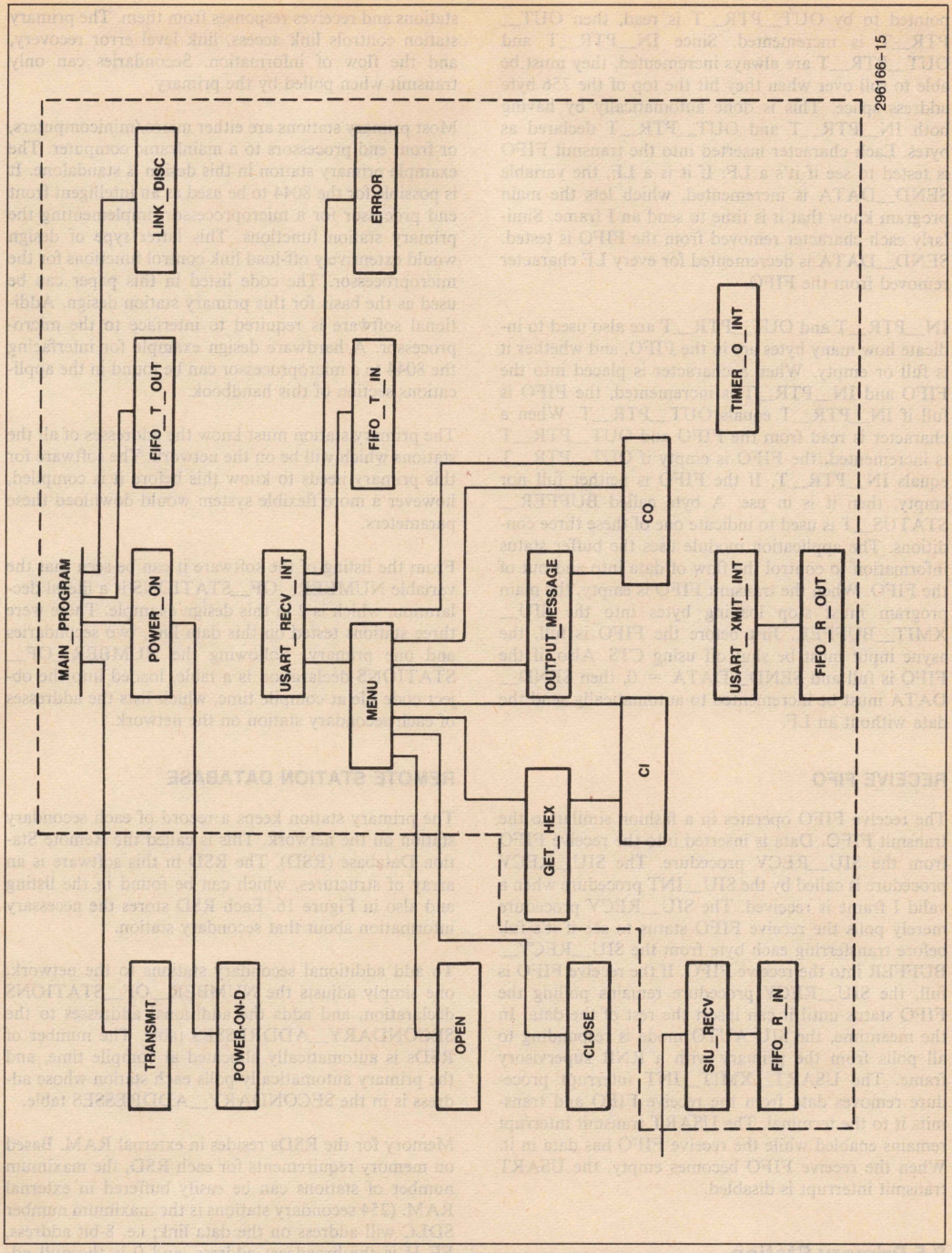


Figure 15. Block Diagram of User Software

pointed to by OUT_PTR_T is read, then OUT_PTR_T is incremented. Since IN_PTR_T and OUT_PTR_T are always incremented, they must be able to roll over when they hit the top of the 256 byte address space. This is done automatically by having both IN_PTR_T and OUT_PTR_T declared as bytes. Each character inserted into the transmit FIFO is tested to see if it's a LF. If it is a LF, the variable SEND_DATA is incremented, which lets the main program know that it is time to send an I frame. Similarly each character removed from the FIFO is tested. SEND_DATA is decremented for every LF character removed from the FIFO.

IN_PTR_T and OUT_PTR_T are also used to indicate how many bytes are in the FIFO, and whether it is full or empty. When a character is placed into the FIFO and IN_PTR_T is incremented, the FIFO is full if IN_PTR_T equals OUT_PTR_T. When a character is read from the FIFO and OUT_PTR_T is incremented, the FIFO is empty if OUT_PTR_T equals IN_PTR_T. If the FIFO is neither full nor empty, then it is in use. A byte called BUFFER_STATUS_T is used to indicate one of these three conditions. The application module uses the buffer status information to control the flow of data into and out of the FIFO. When the transmit FIFO is empty, the main program must stop loading bytes into the SIU_XMIT_BUFFER. Just before the FIFO is full, the async input must be shut off using CTS. Also, if the FIFO is full and SEND_DATA = 0, then SEND_DATA must be incremented to automatically send the data without an LF.

RECEIVE FIFO

The receive FIFO operates in a fashion similar to the transmit FIFO. Data is inserted into the receive FIFO from the SIU_RECV procedure. The SIU_RECV procedure is called by the SIU_INT procedure when a valid I frame is received. The SIU_RECV procedure merely polls the receive FIFO status to see if it's full before transferring each byte from the SIU_RECV_BUFFER into the receive FIFO. If the receive FIFO is full, the SIU_RECV procedure remains polling the FIFO status until it can insert the rest of the data. In the meantime, the SIU AUTO mode is responding to all polls from the primary with a RNR supervisory frame. The USART_XMIT_INT interrupt procedure removes data from the receive FIFO and transmits it to the terminal. The USART transmit interrupt remains enabled while the receive FIFO has data in it. When the receive FIFO becomes empty, the USART transmit interrupt is disabled.

2.5 Primary Station

The primary station is responsible for controlling the data link. It issues commands to the secondary stations and receives responses from them. The primary station controls link access, link level error recovery, and the flow of information. Secondaries can only transmit when polled by the primary.

Most primary stations are either micro/minicomputers, or front end processors to a mainframe computer. The example primary station in this design is standalone. It is possible for the 8044 to be used as an intelligent front end processor for a microprocessor, implementing the primary station functions. This latter type of design would extensively off-load link control functions for the microprocessor. The code listed in this paper can be used as the basis for this primary station design. Additional software is required to interface to the microprocessor. A hardware design example for interfacing the 8044 to a microprocessor can be found in the applications section of this handbook.

The primary station must know the addresses of all the stations which will be on the network. The software for this primary needs to know this before it is compiled, however a more flexible system would download these parameters.

From the listing of the software it can be seen that the variable NUMBER_OF_STATIONS is a literal declaration, which is 2 in this design example. There were three stations tested on this data link, two secondaries and one primary. Following the NUMBER_OF_STATIONS declaration is a table, loaded into the object code file at compile time, which lists the addresses of each secondary station on the network.

REMOTE STATION DATABASE

The primary station keeps a record of each secondary station on the network. This is called the Remote Station Database (RSD). The RSD in this software is an array of structures, which can be found in the listing and also in Figure 16. Each RSD stores the necessary information about that secondary station.

To add additional secondary stations to the network, one simply adjusts the NUMBER_OF_STATIONS declaration, and adds the additional addresses to the SECONDARY_ADDRESSES table. The number of RSDs is automatically allocated at compile time, and the primary automatically polls each station whose address is in the SECONDARY_ADDRESSES table.

Memory for the RSDs resides in external RAM. Based on memory requirements for each RSD, the maximum number of stations can be easily buffered in external RAM. (254 secondary stations is the maximum number SDLC will address on the data link; i.e. 8-bit address, FF H is the broadcast address, and 0 is the null address. Each RSD uses 70 bytes of RAM. 70 x 254 = 17,780.)

The station state, in the RSD structure, maintains the status of the secondary. If this byte indicates that the secondary is in the DISCONNECT_S, then the primary tries to put the station in the I_T_S by sending an SNRM. If the response is a UA then the station state changes into the I_T_S. Any other frame received results in the station state remaining in the DISCONNECT_S. When the RSD indicates that the station state is in the I_T_S, the primary will send either an I, RR, or RNR command, depending on the local and remote buffer status. When the station state equals GO_TO_DISC the primary will send a DISC command. If the response is a UA frame, the station state will change to DISCONNECT_S, else the station state will remain in GO_TO_DISC. The station state is set to GO_TO_DISC when one of the following responses occur:

1. A receive buffer overrun in the primary.
2. An I frame is received and Nr(P) ≠ Ns(S).
3. An I frame or a Supervisory frame is received and Ns(P) + 1 ≠ Nr(S) and Ns(P) ≠ Nr(S).
4. A FRMR response is received.
5. An RD response is received.
6. An unknown response is received.

The send count (Ns) and receive count (Nr) are also maintained in the RSD. Each time an I frame is sent by the primary and acknowledged by the secondary, Ns is incremented. Nr is incremented each time a valid I frame is received. BUFFER_STATUS indicates the status of the secondary station's buffer. If an RR response is received, BUFFER_STATUS is set to BUFFER_READY. If a RNR response is received, BUFFER_STATUS is set to BUFFER_NOT_READY.

BUFFERING

The buffering for the primary station is as follows: within each RSD is a 64 byte array buffer which is initially empty. When the primary receives an I frame, it looks for a match between the first byte of the I frame and the addresses of the secondaries on the network. If a match exists, the primary places the data in the RSD buffer of the destination station. The INFO_LENGTH in the RSD indicates how many bytes are in the buffer. If INFO_LENGTH equals 0, then the buffer is empty. The primary can buffer only one I frame per station. If a second I frame is received while the addressed secondary's RSD buffer is full, the primary cannot receive any more I frames. At this point the primary continues to poll the secondaries using RNR supervisory frame.

PRIMARY STATION SOFTWARE

A block diagram of the primary station software is shown in Figure 17. The primary station software consists of a main program, one interrupt routine, and several procedures. The POWER_ON procedure begins by initializing the SIU's DMA and enabling the receiver. Then each RSD is initialized. The DPLL and the timers are set, and finally the TIMER 0 interrupt is enabled.

The main program consists of an iterative do loop within a do forever loop. The iterative do loop polls each secondary station once through the do loop. The variable STATION_NUMBER is the counter for the iterative do statement which is also used as an index to the array of RSD structures. The primary station issues one command and receives one response from every secondary station each time through the loop. The first statement in the loop loads the secondary station address, indexed by STATION_NUMBER into the array of the RSD structures. Now when the primary sends a command, it will have the secondary's address in the address field of the frame. The automatic address recognition feature is used by the primary to recognize the response from the secondary.

Next, the main program determines the secondary station's state. Based on this state, the primary knows what command to send. If the station is in the DISCONNECT_S, the primary calls the SNRM_P procedure to try and put the secondary in the I_T_S. If the station state is in the GO_TO_DISC state, the DISC_P is called to try and put the secondary in the L_D_S. If the secondary is in neither one of the above two states, then it is in the I_T_S. When the secondary is in the I_T_S, the primary could send one of three commands: I, RR, or RNR. If the RSD's buffer has data in it, indicated by INFO_LENGTH being greater than zero, and the secondary's BUFFER_STATUS equals BUFFER_READY, then an I frame will be sent. Else if RPB = 0, an RR supervisory frame will be sent. If neither one of these cases is true, then an RNR will be sent. The last statement in the main program checks the RPB bit. If set to one, the BUFFER_TRANSFER procedure is called, which transfers the data from the SIU receive buffer to the appropriate RSD buffer.

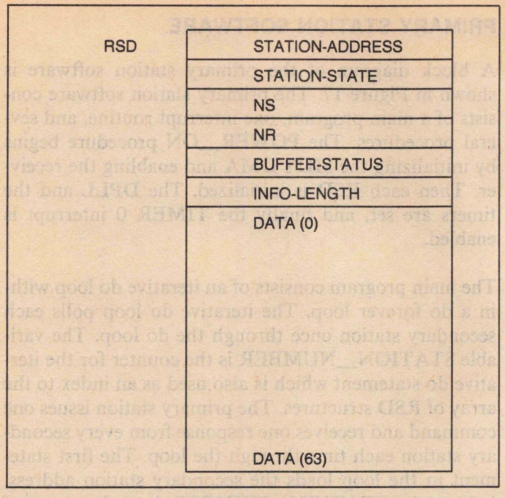


Figure 16. Remote Station Database Structure

RECEIVE TIME OUT

Each time a frame is transmitted, the primary sets a receive time out timer; Timer 0. If a response is not received within a certain time, the primary returns to the main program and continues polling the rest of the stations. The minimum length of time the primary should wait for a response can be calculated as the sum of the following parameters.

1. Propagation time to the secondary station
2. Clear-to-send at the secondary station's DCE
3. Appropriate time for secondary station processing
4. Propagation time from the secondary station
5. Maximum frame length time

The clear-to-send time and the propagation time are negligible for a local network at low bit rates. However, the turnaround time and the maximum frame length time are significant factors. Using the 8044 secondaries in the AUTO mode minimizes turnaround time. The maximum frame length time comes from the fact the 8044 does not generate an interrupt from a received frame until it has been completely received, and the CRC is verified as correct. This means that the time-out is bit rate dependent.

Ns AND Nr CHECK PROCEDURES

Each time an I frame or supervisory frame is received, the Nr field in the control byte must be checked. Since this data link only allows one outstanding frame, a valid Nr would satisfy either one of two equations; Ns(P) + 1 = Nr(S) the I frame previously sent by the primary is acknowledged, Ns(P) = Nr(S) the I frame previously sent is not acknowledged. If either one of these two cases is true, the CHECK_NR procedure returns a parameter of TRUE; otherwise a FALSE parameter is returned. If an acknowledgement is received, the Ns byte in the RSD structure is incremented, and the Information buffer may be cleared. Otherwise the information buffer remains full.

When an I frame is received, the Ns field has to be checked also. If Nr(P) = Ns(S), then the procedure returns TRUE, otherwise a FALSE is returned.

RECEIVE PROCEDURE

The receive procedure is called when a supervisory or information frame is sent, and a response is received before the time-out period. The RECEIVE procedure can be broken down into three parts. The first part is entered if an I frame is received. When an I frame is received, Ns, Nr and buffer overrun are checked. If there is a buffer overrun, or there is an error in either Ns or Nr, then the station state is set to GO_TO_DISC. Otherwise Nr in the RSD is incremented, the receive field length is saved, and the RPB bit is set. By incrementing the Nr field, the I frame just received is acknowledged the next time the primary polls the secondary with an I frame or a supervisory frame. Setting RBP protects the received data, and also tells the main program that there is data to transfer to one of the RSD buffers.

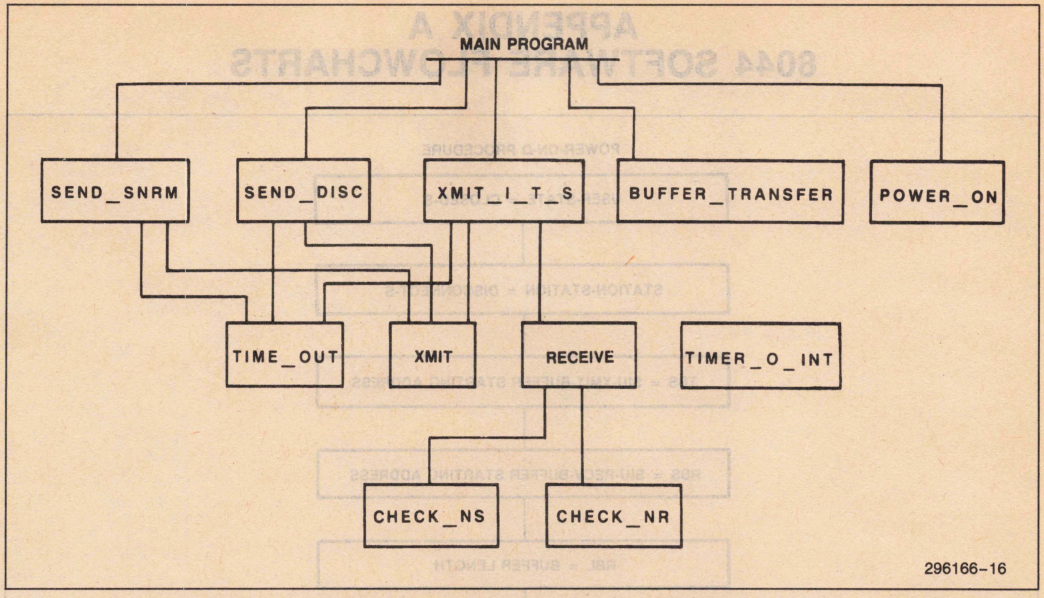


Figure 17. Block Diagram of Primary Station Software Structure

If a supervisory frame is received, the Nr field is checked. If a FALSE is returned, then the station state is set to GO_TO_DISC. If the supervisory frame received was an RNR, buffer status is set to not ready. If the response is not an I frame, nor a supervisory frame, then it must be an Unnumbered frame.

The only Unnumbered frames the primary recognizes are UA, DM, and FRMR. In any event, the station state is set to GO_TO_DISC. However, if the frame received is a FRMR, Nr in the second data byte of the I field is checked to see if the secondary acknowledged an I frame received before it went into the FRMR state. If this is not done and the secondary acknowledged an I frame which the primary did not recognize, the primary transmits the I frame when the secondary returns to the I_T_S. In this case, the secondary would receive duplicate I frames.

APPENDIX A
8044 SOFTWARE FLOWCHARTS

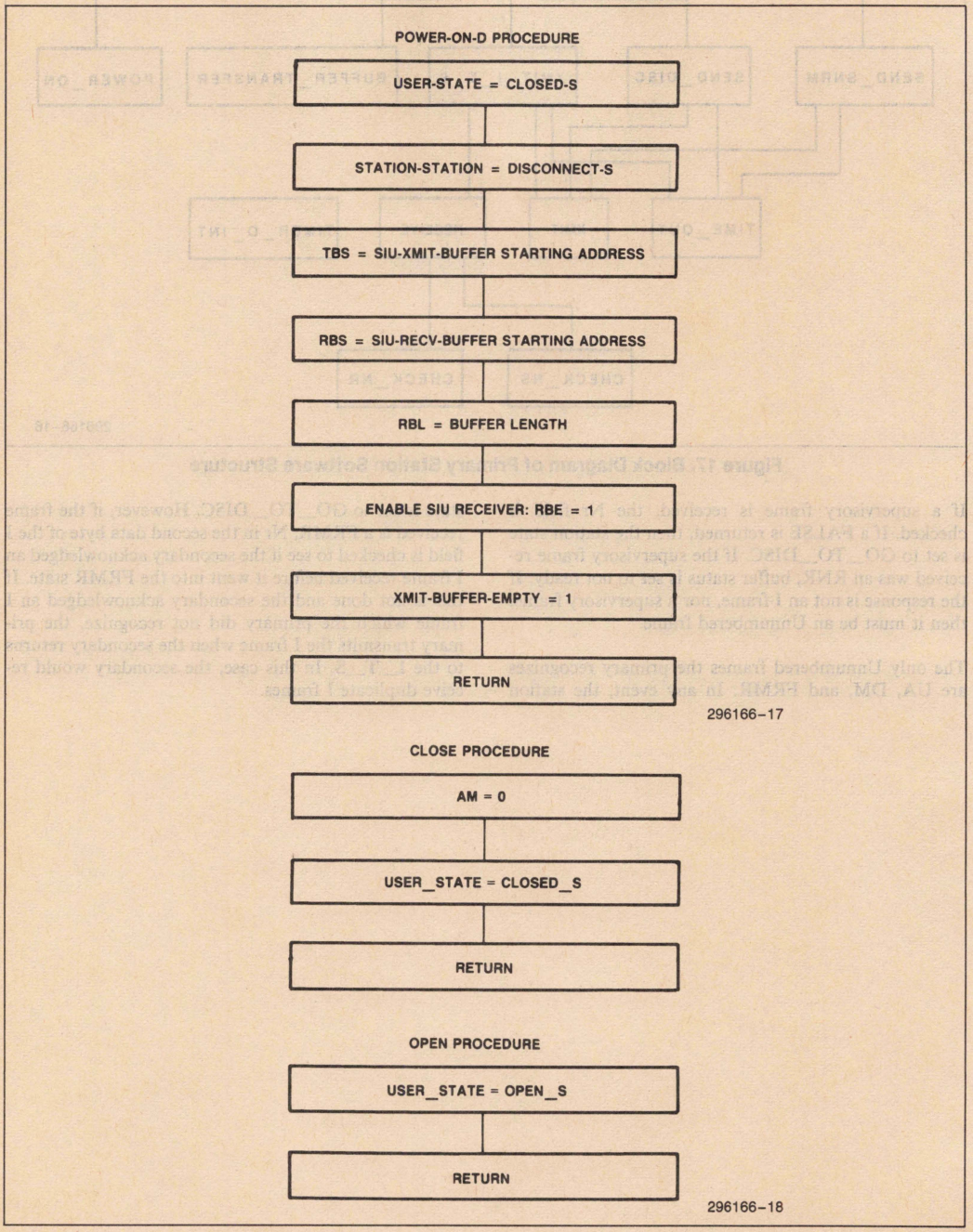


Figure 18. Secondary Station Driver Flow Chart

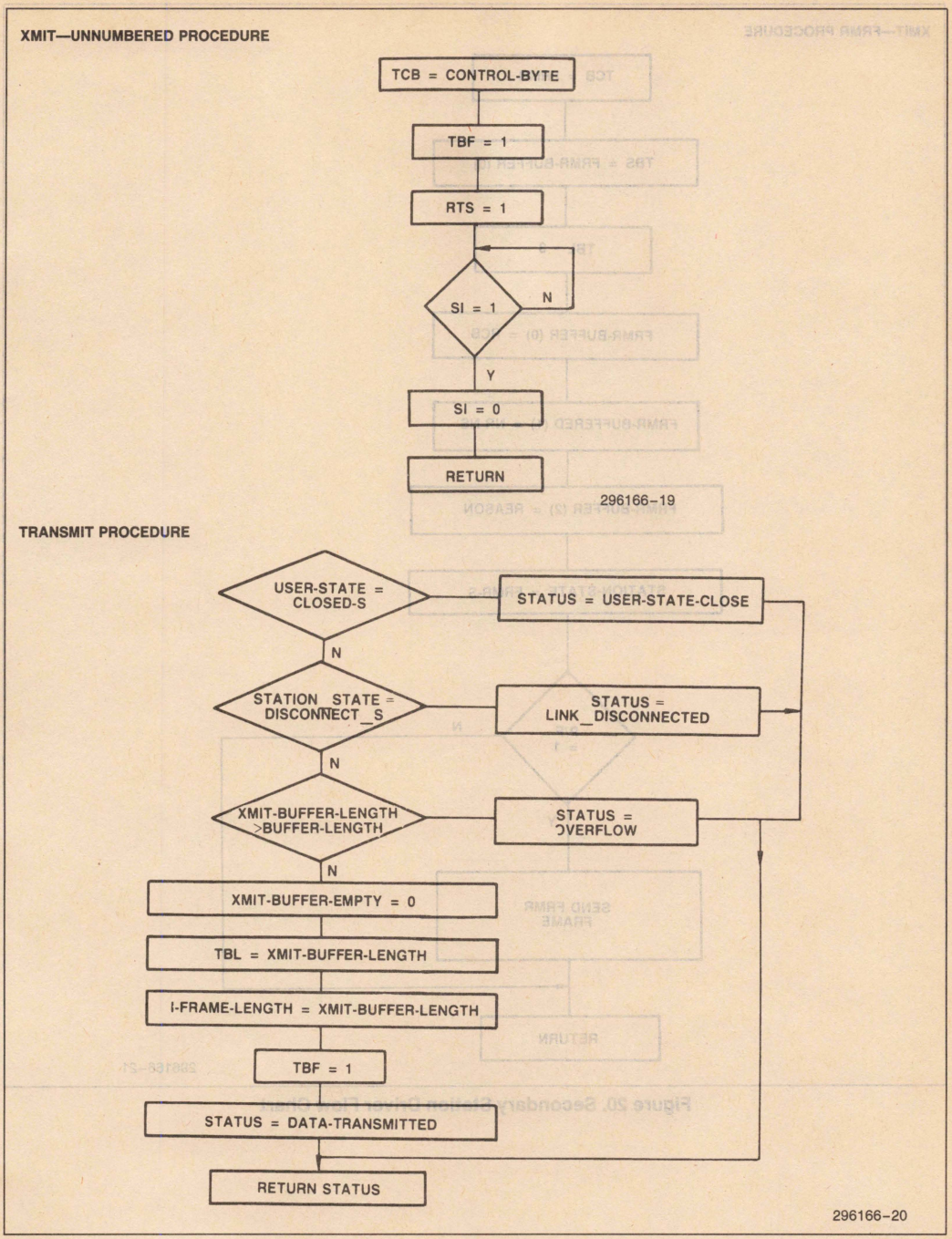


Figure 19. Secondary Station Driver Flow Chart

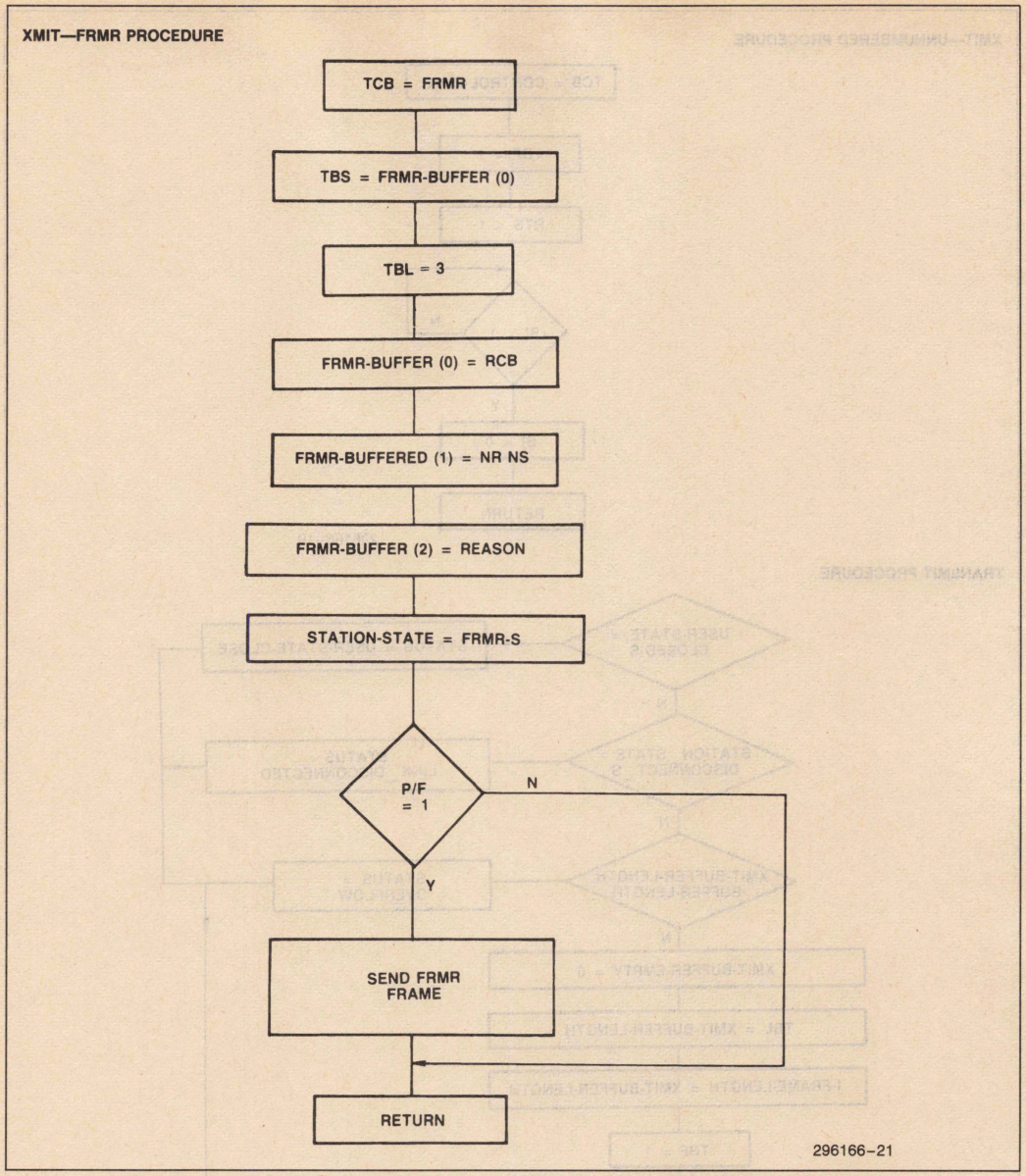


Figure 20. Secondary Station Driver Flow Chart

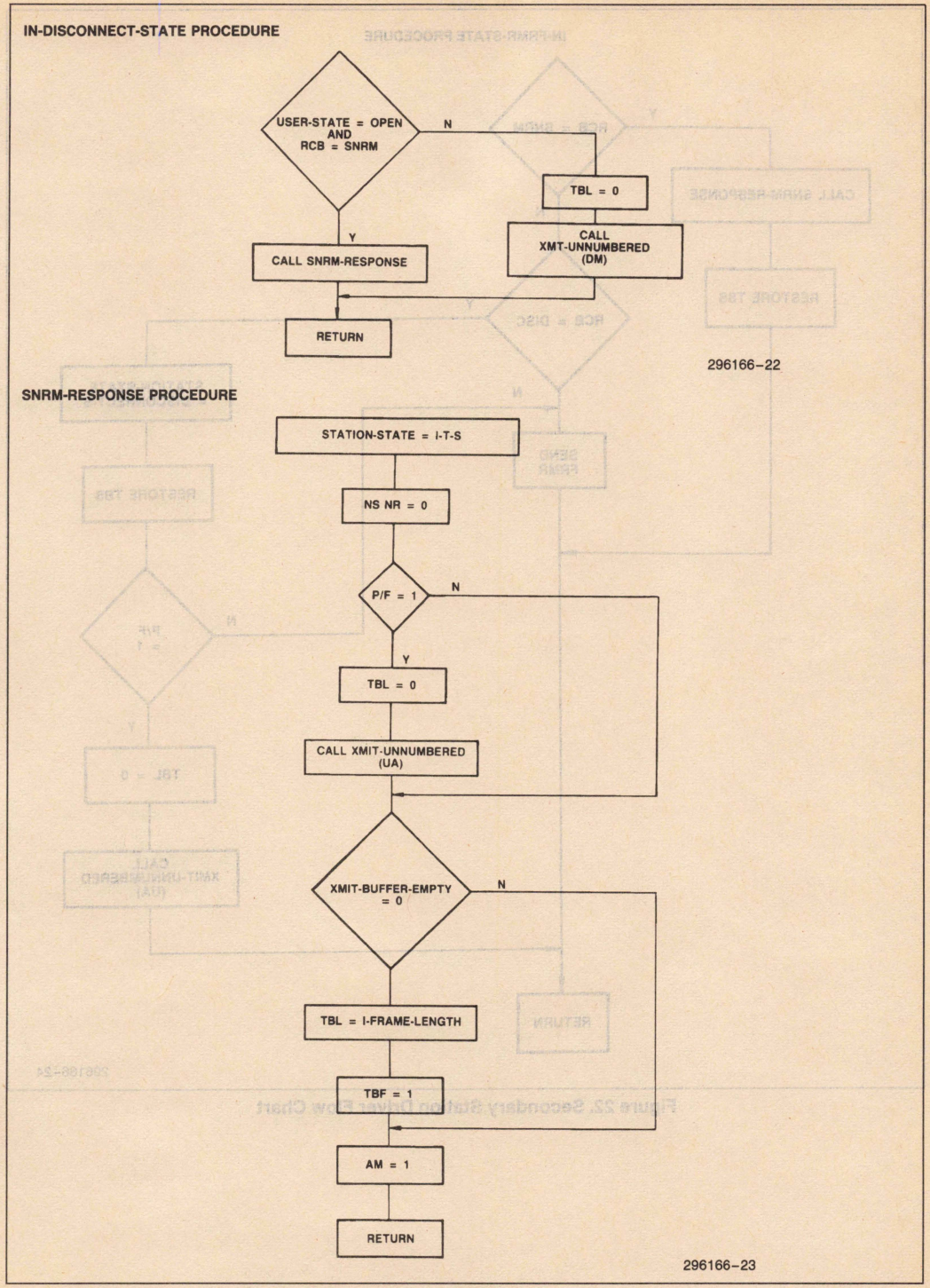


Figure 21. Secondary Station Driver Flow Chart

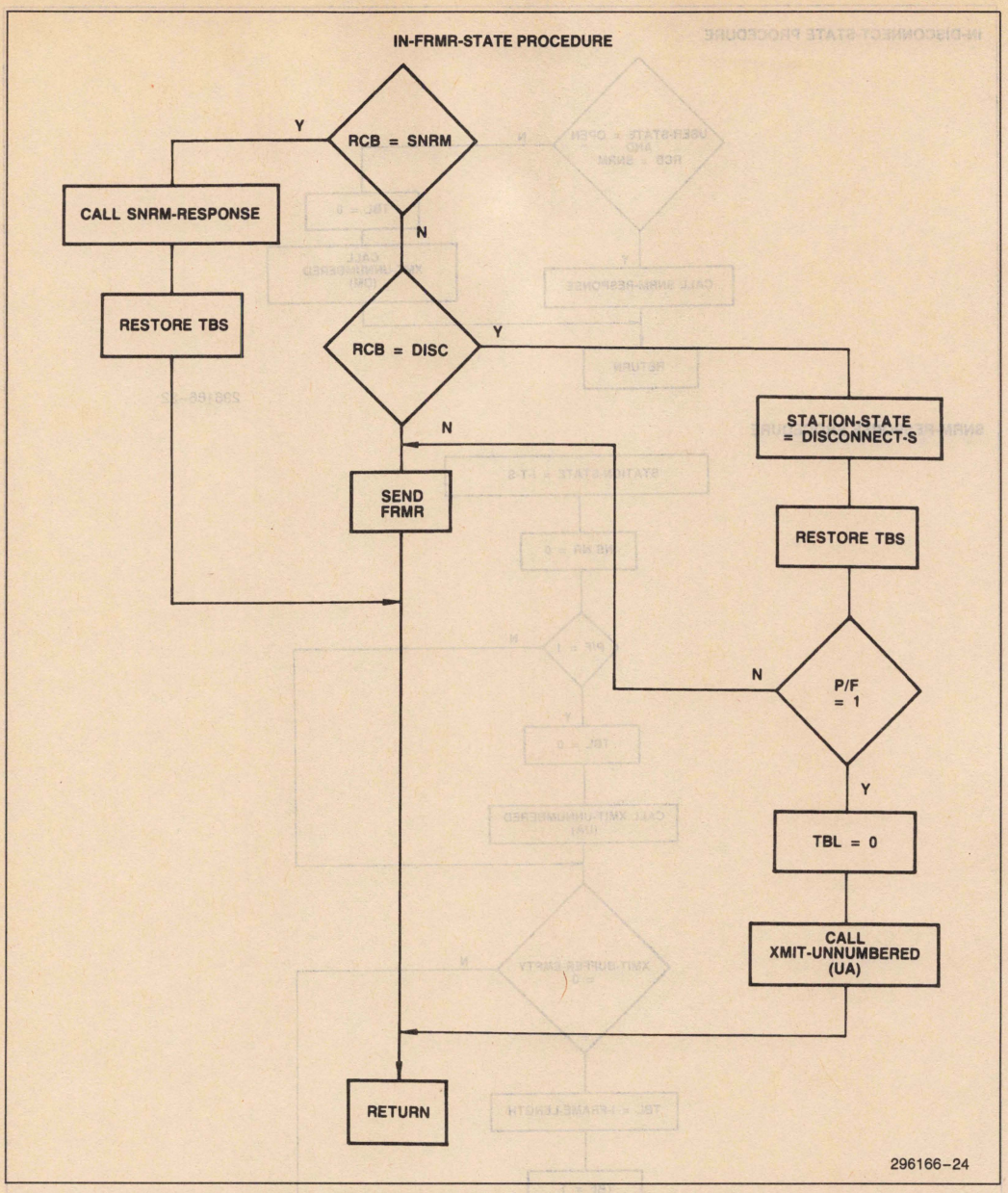


Figure 22. Secondary Station Driver Flow Chart

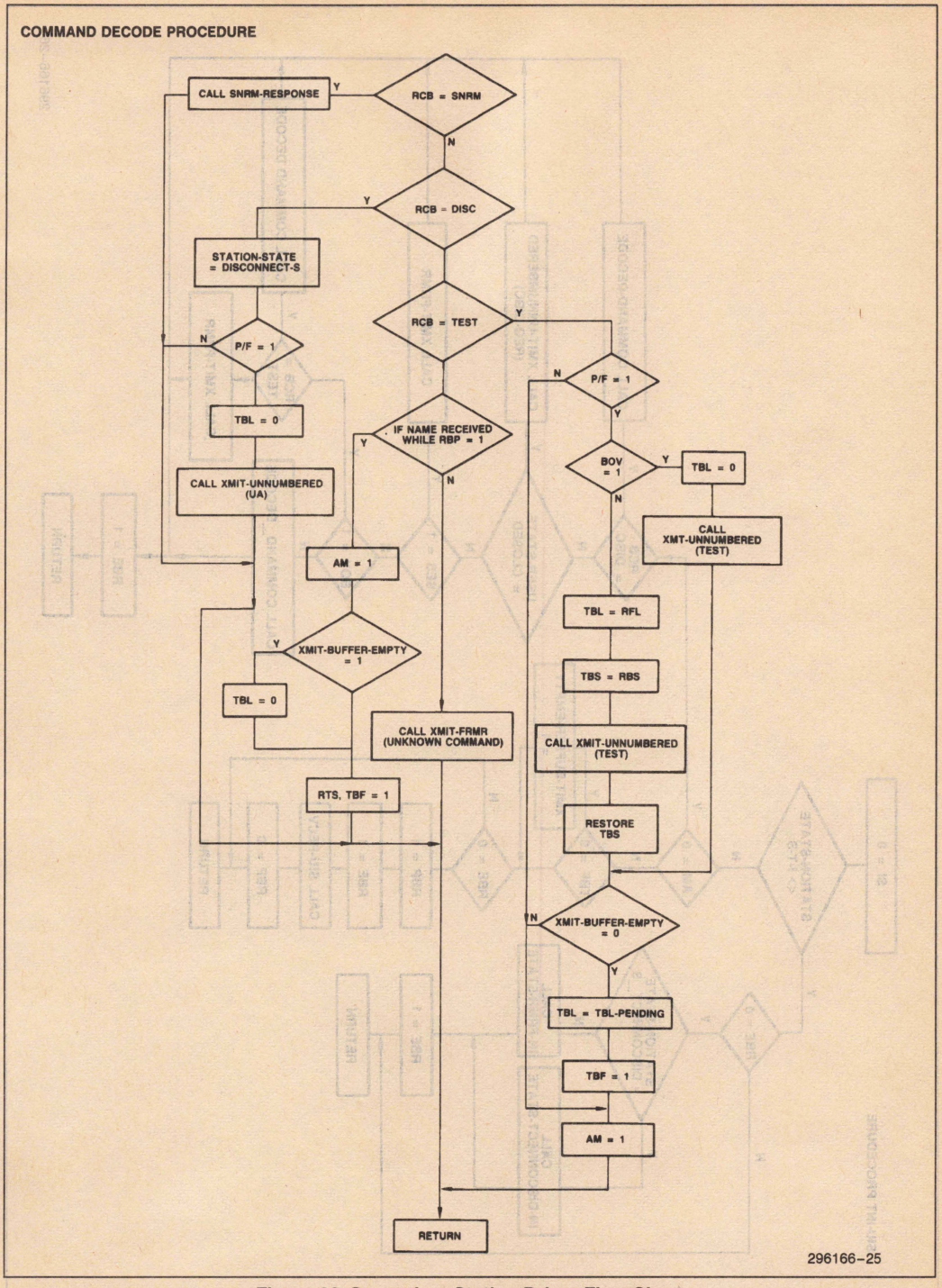


Figure 23. Secondary Station Driver Flow Chart

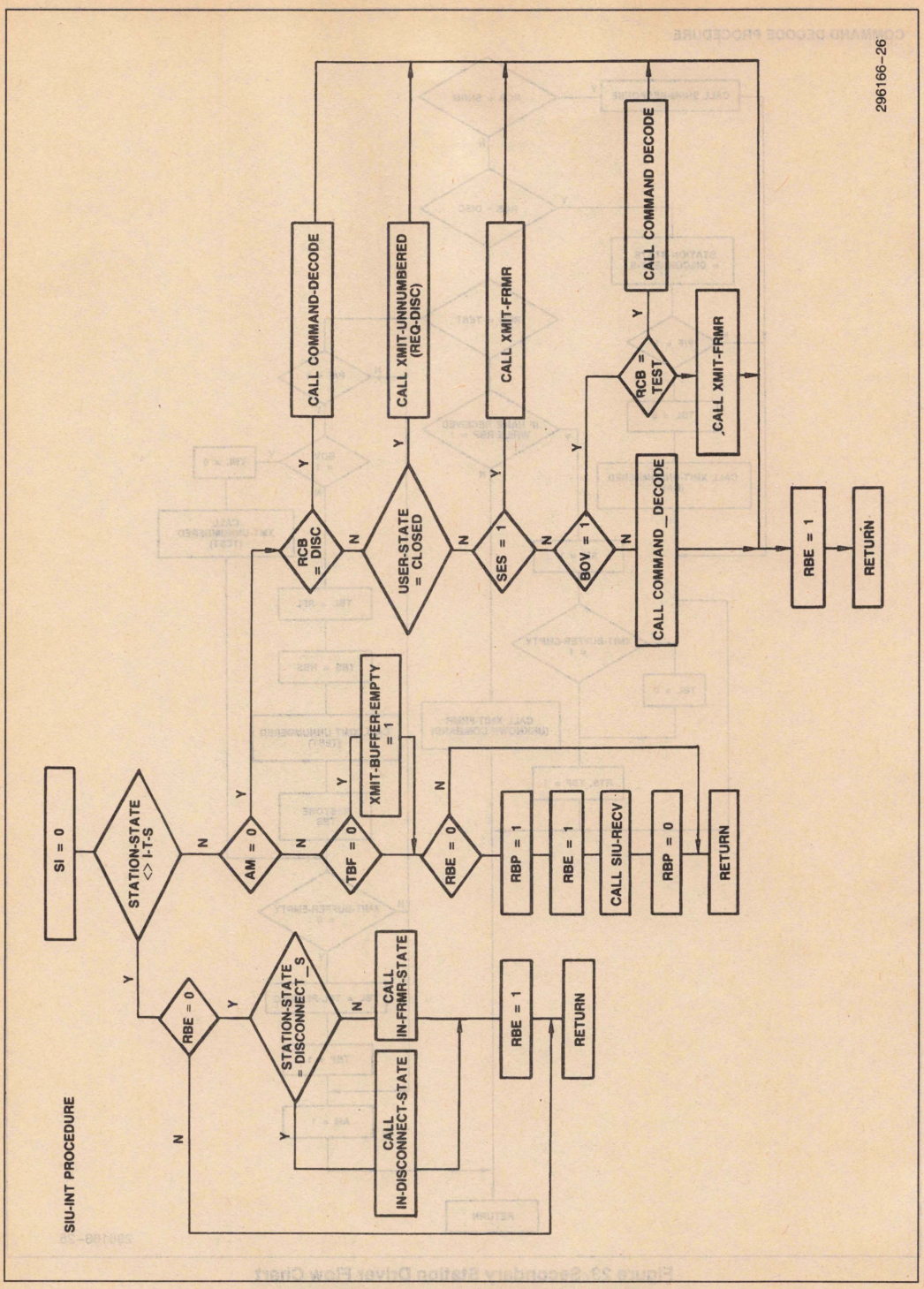


Figure 24. Secondary Station Driver Flow Chart

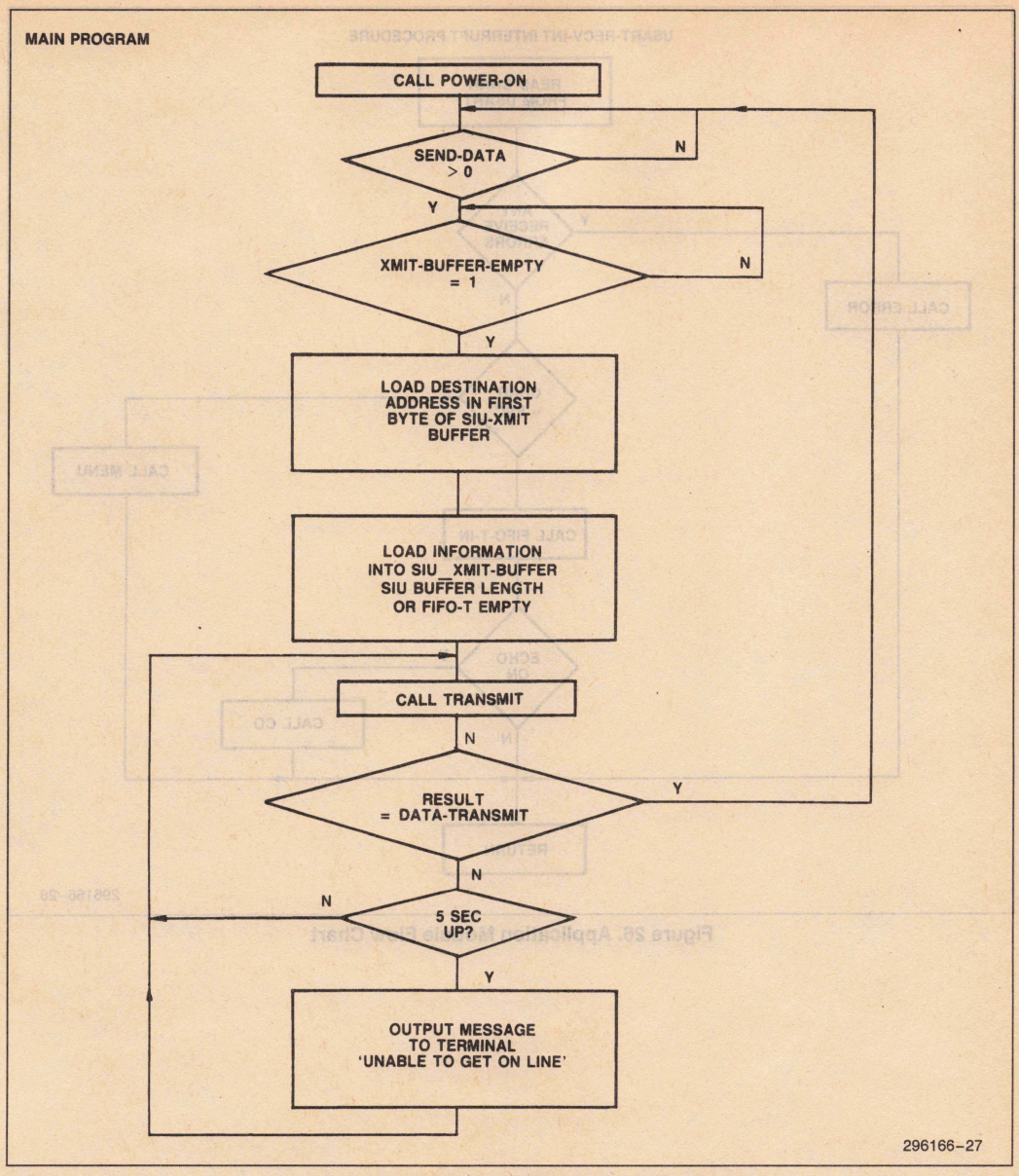


Figure 25. Application Module Flow Chart

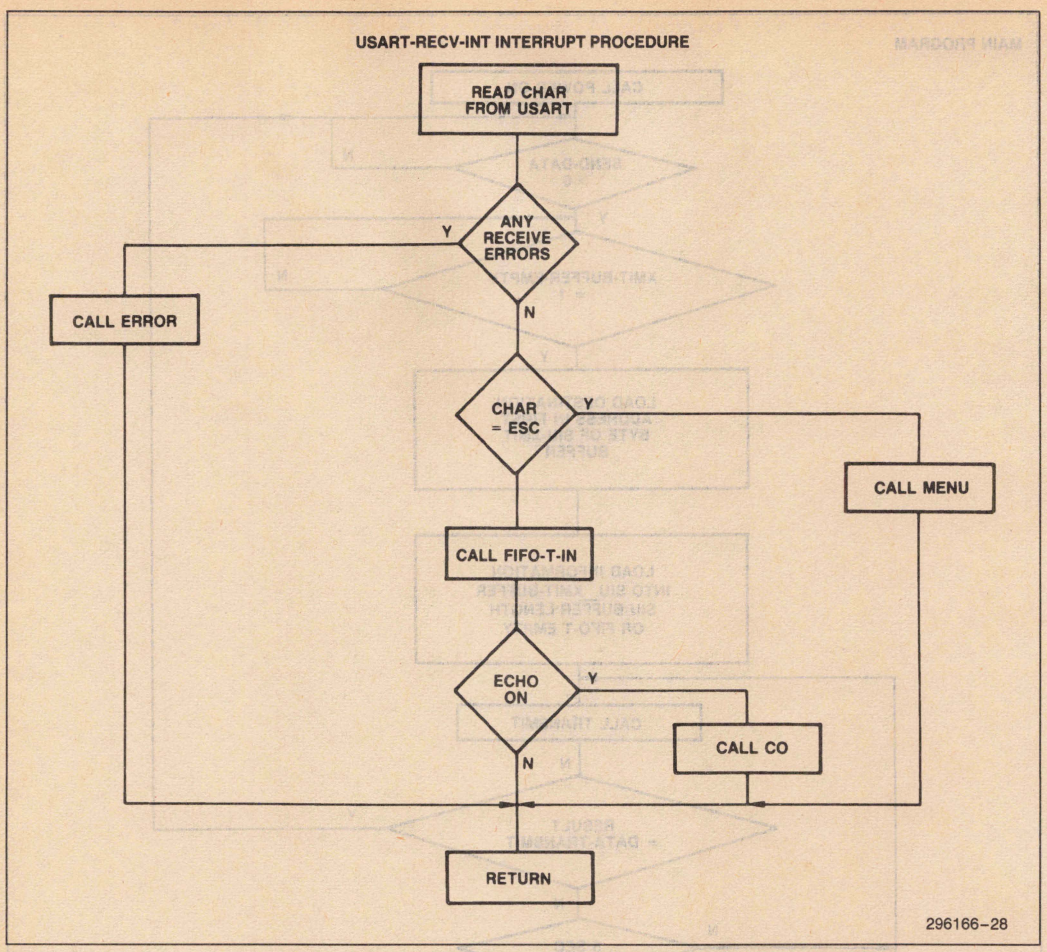


Figure 26. Application Module Flow Chart

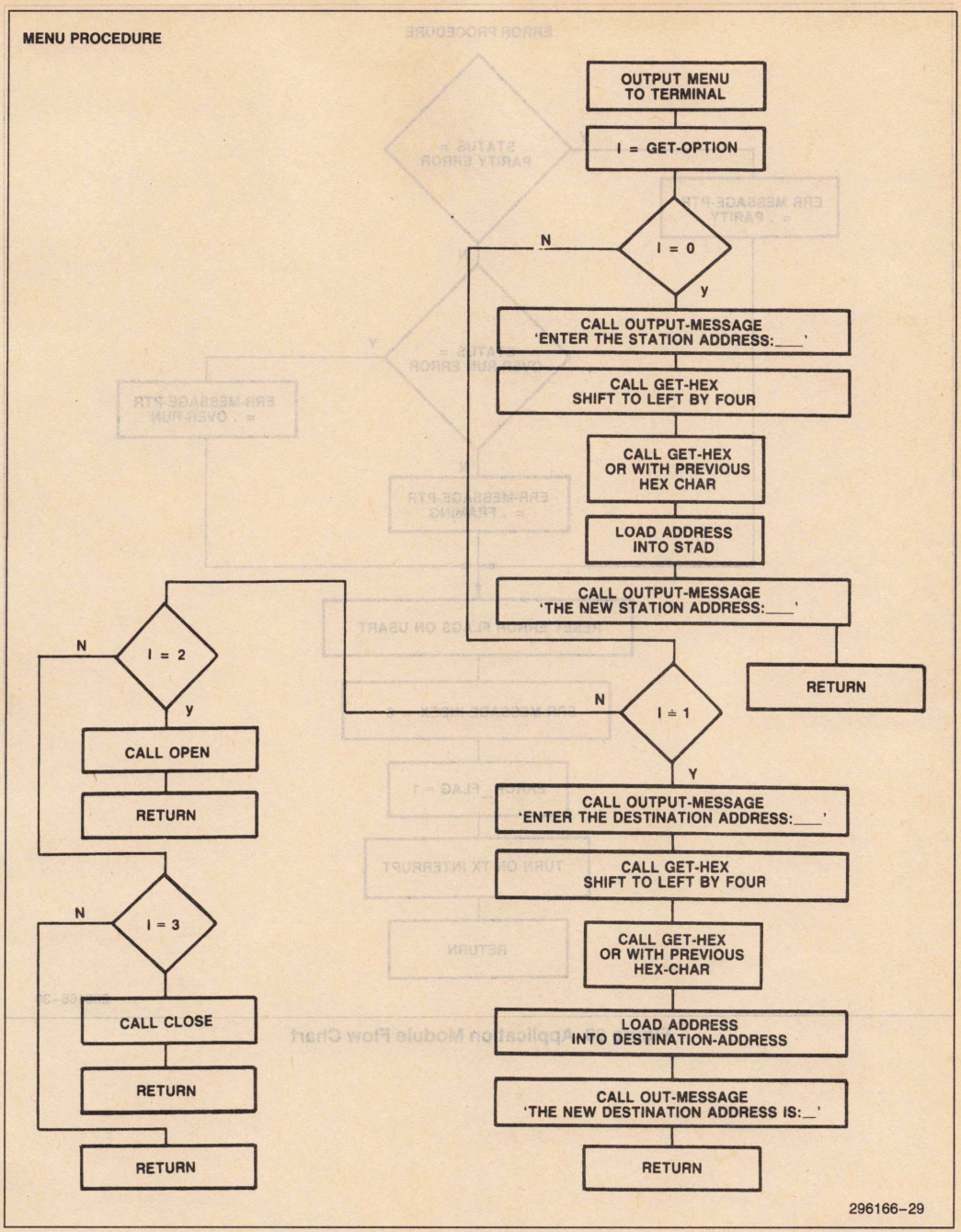


Figure 27. Application Module Flow Chart

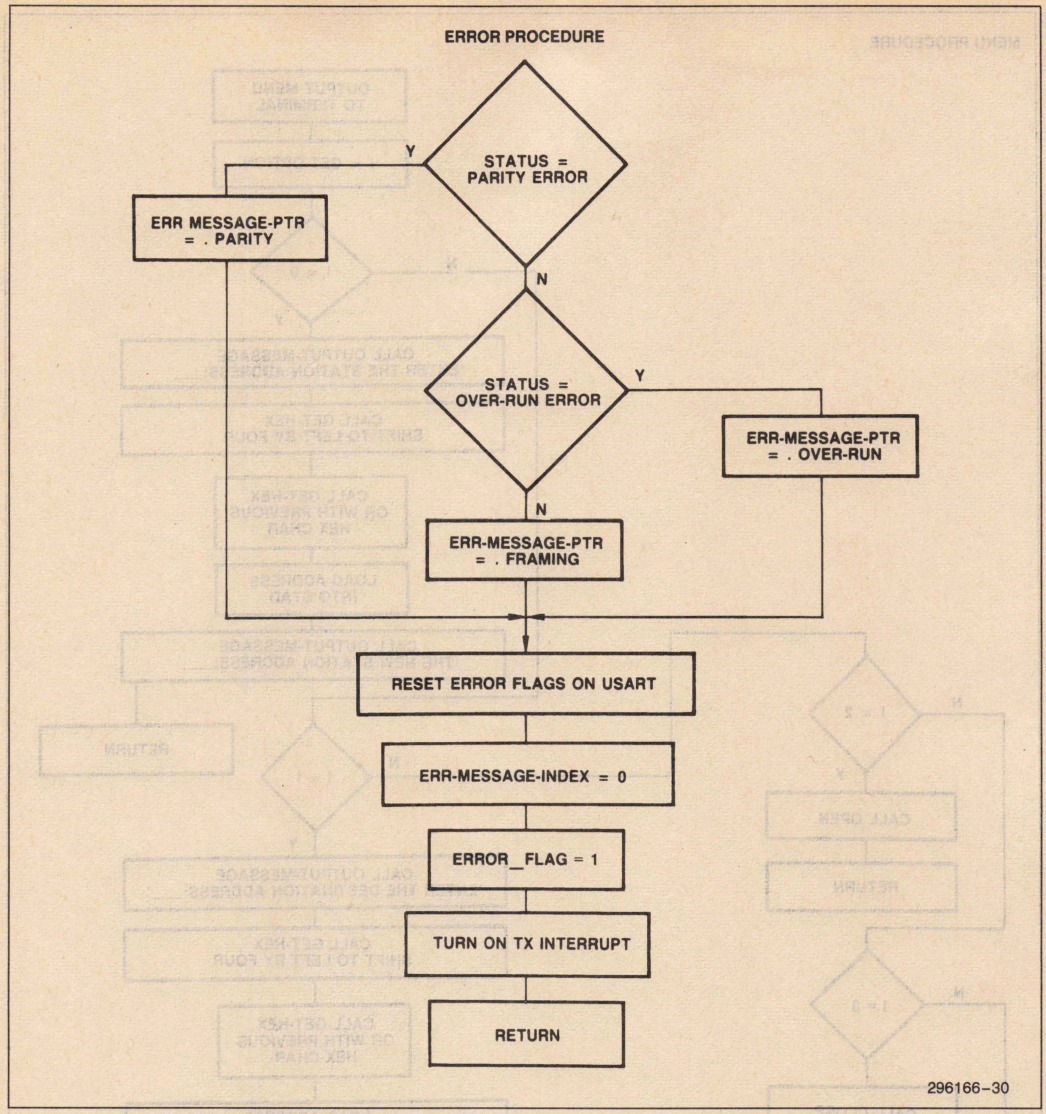


Figure 28. Application Module Flow Chart

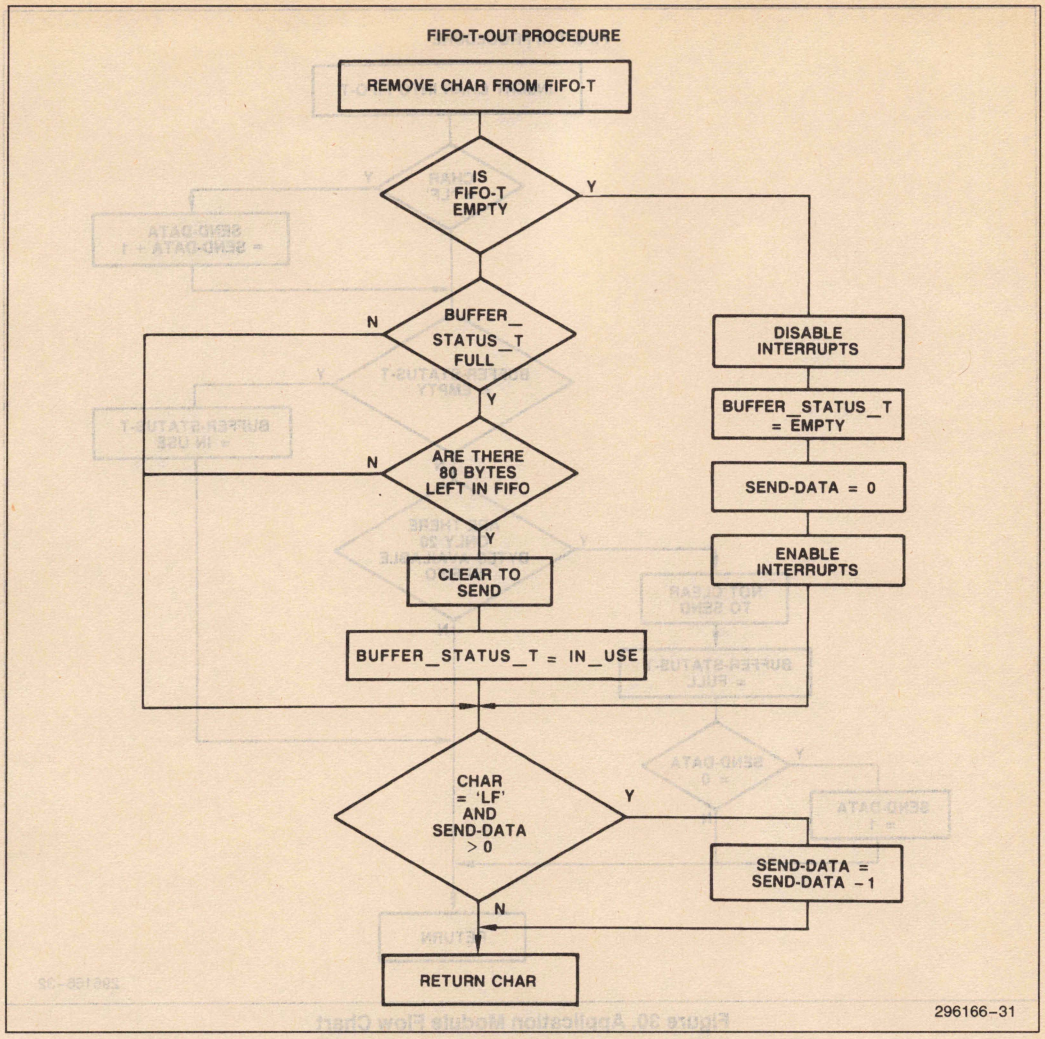


Figure 29. Application Module Flow Chart

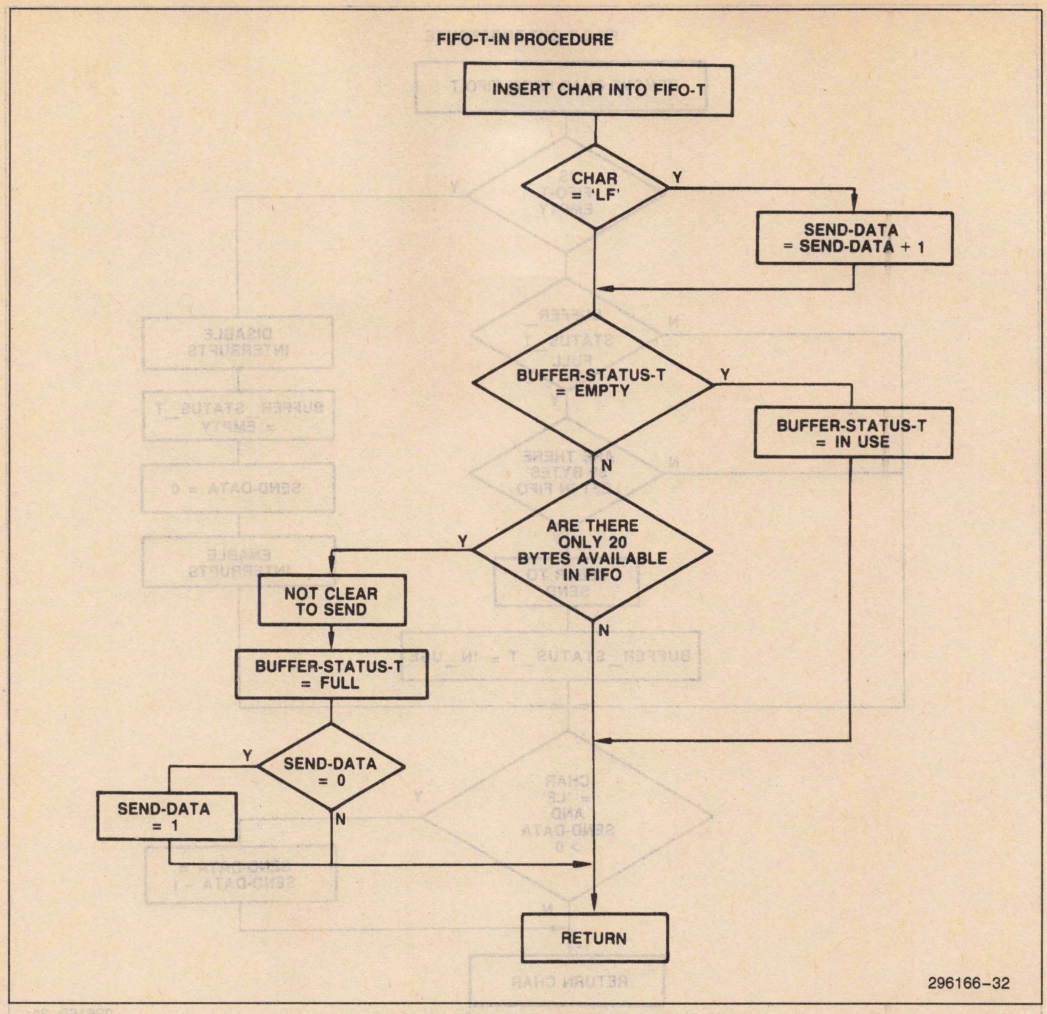


Figure 30. Application Module Flow Chart

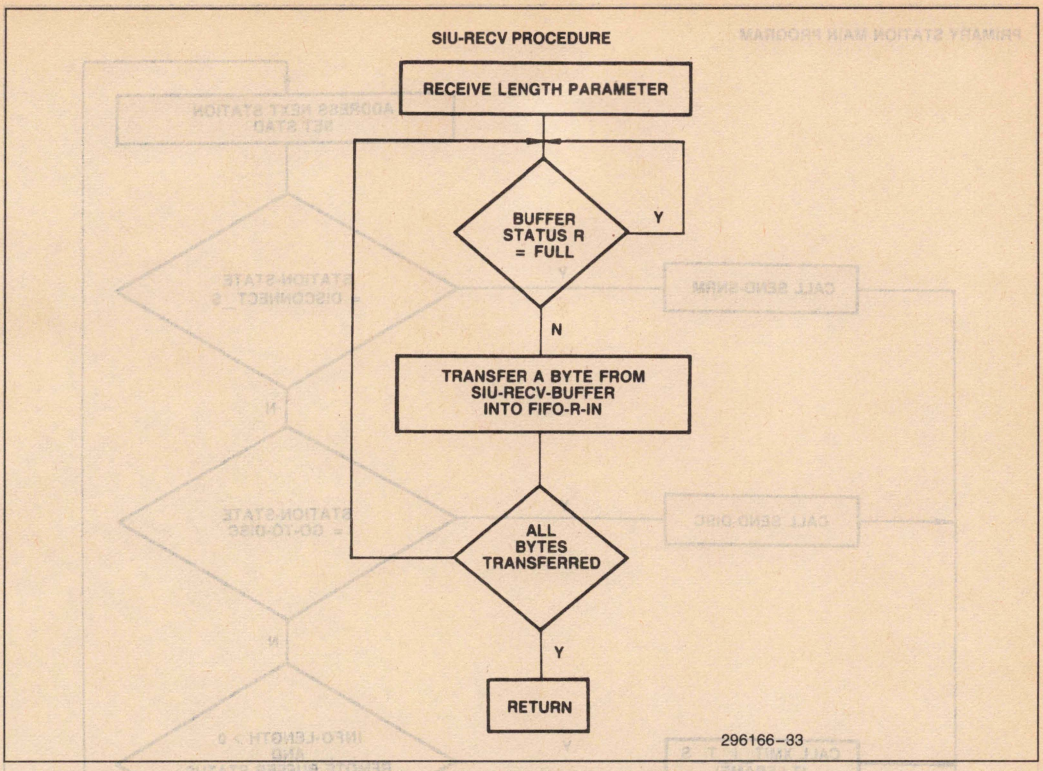


Figure 31. Application Module Flow Chart

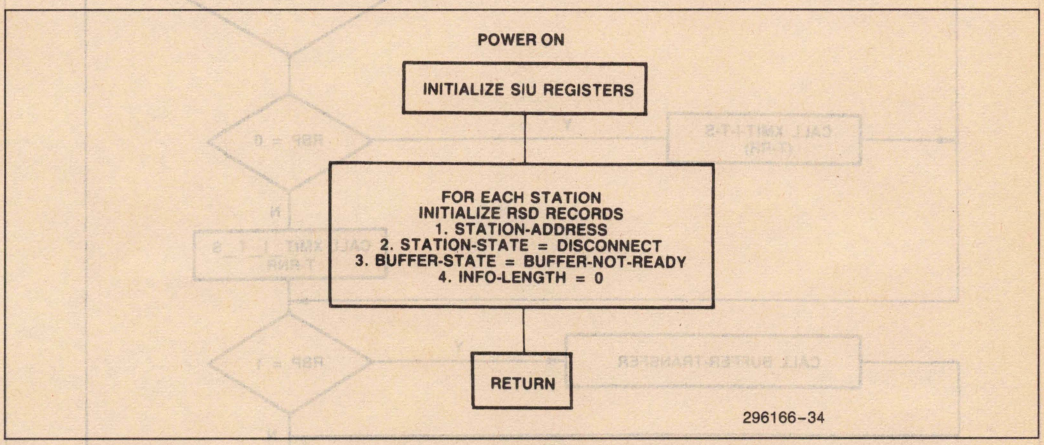


Figure 32. Primary Station Flow Charts

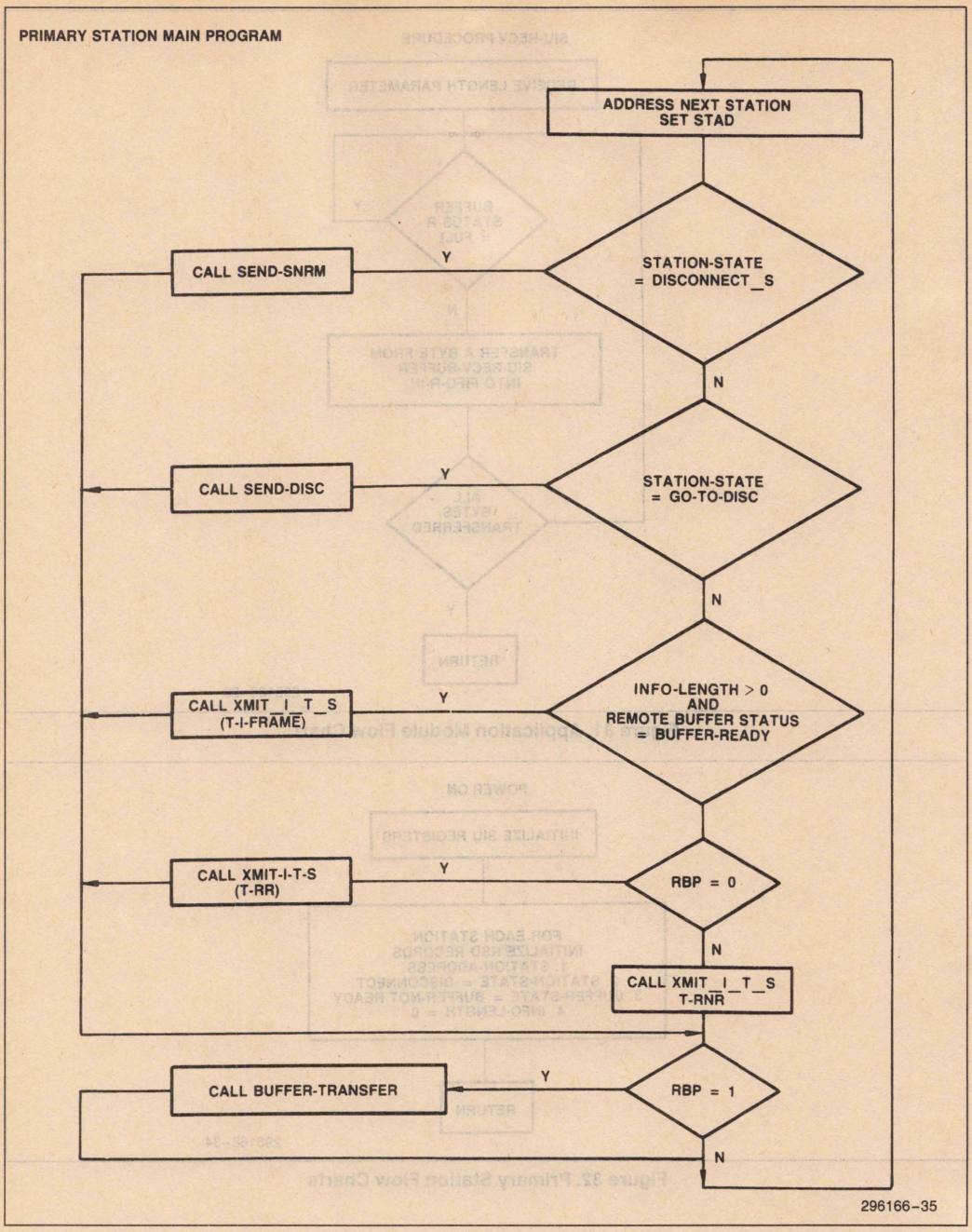


Figure 33. Primary Station Flow Charts

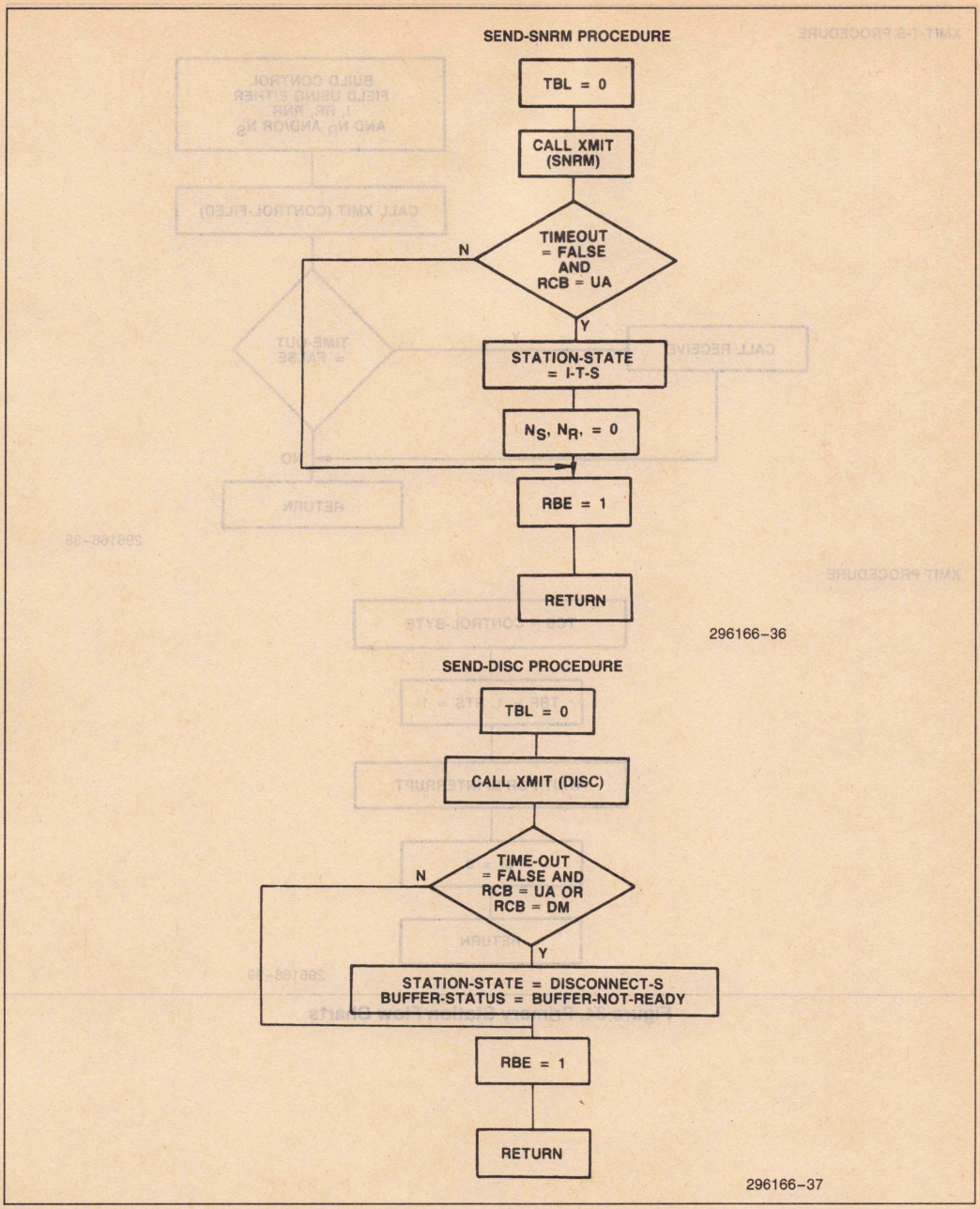


Figure 34. Primary Station Flow Charts

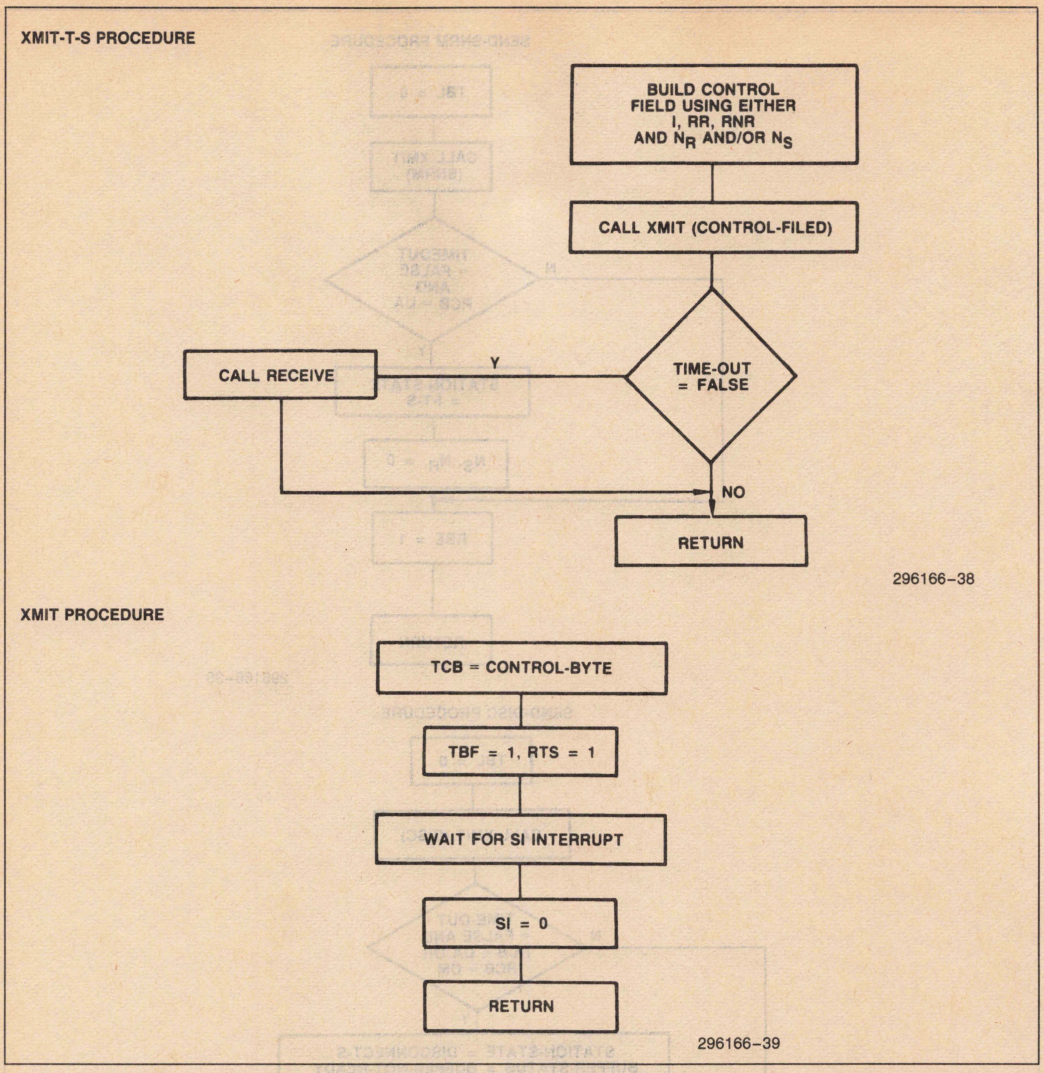


Figure 34. Primary Station Flow Charts

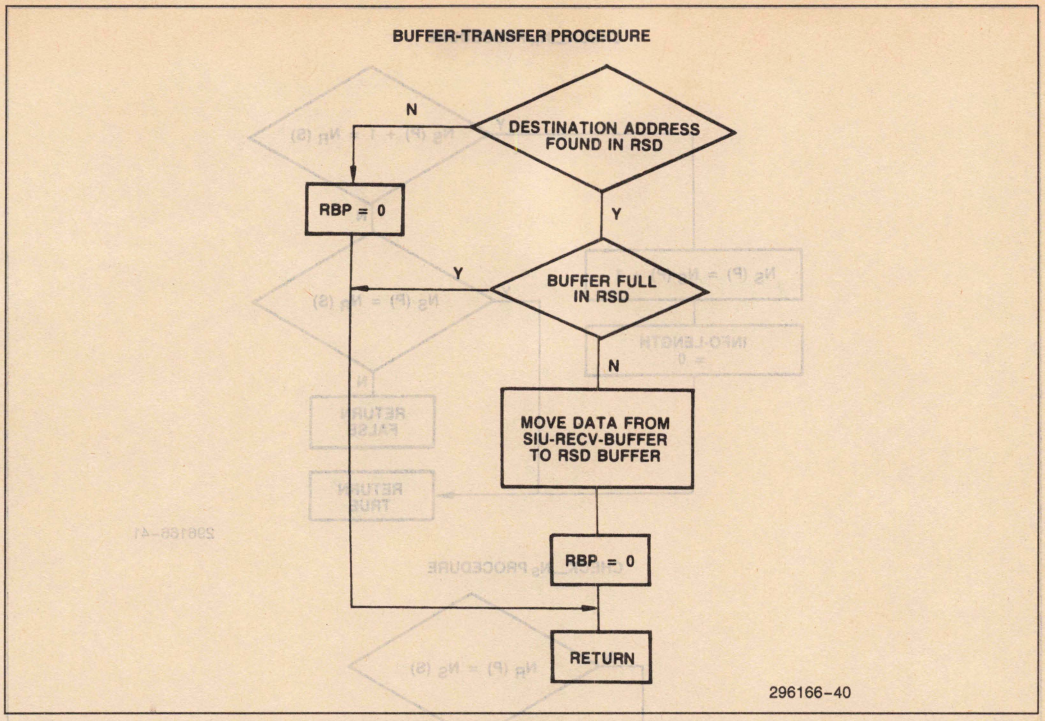


Figure 36. Primary Station Flow Charts

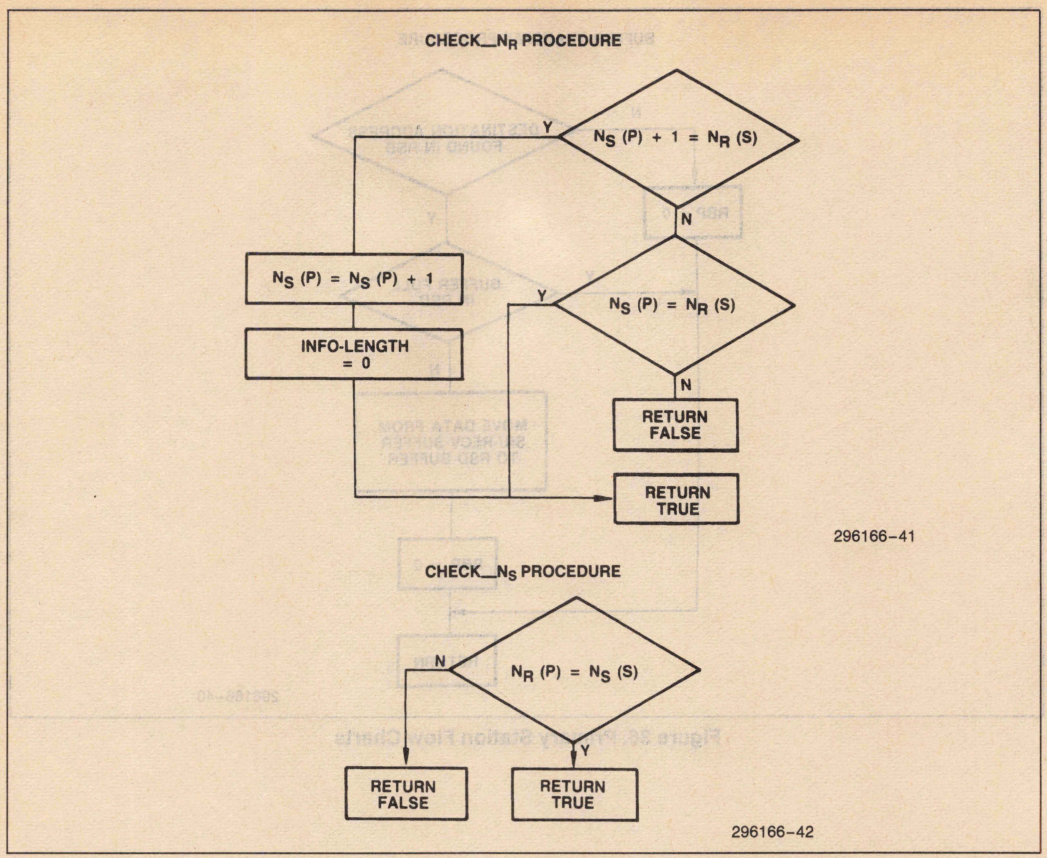


Figure 37. Primary Station Flow Charts

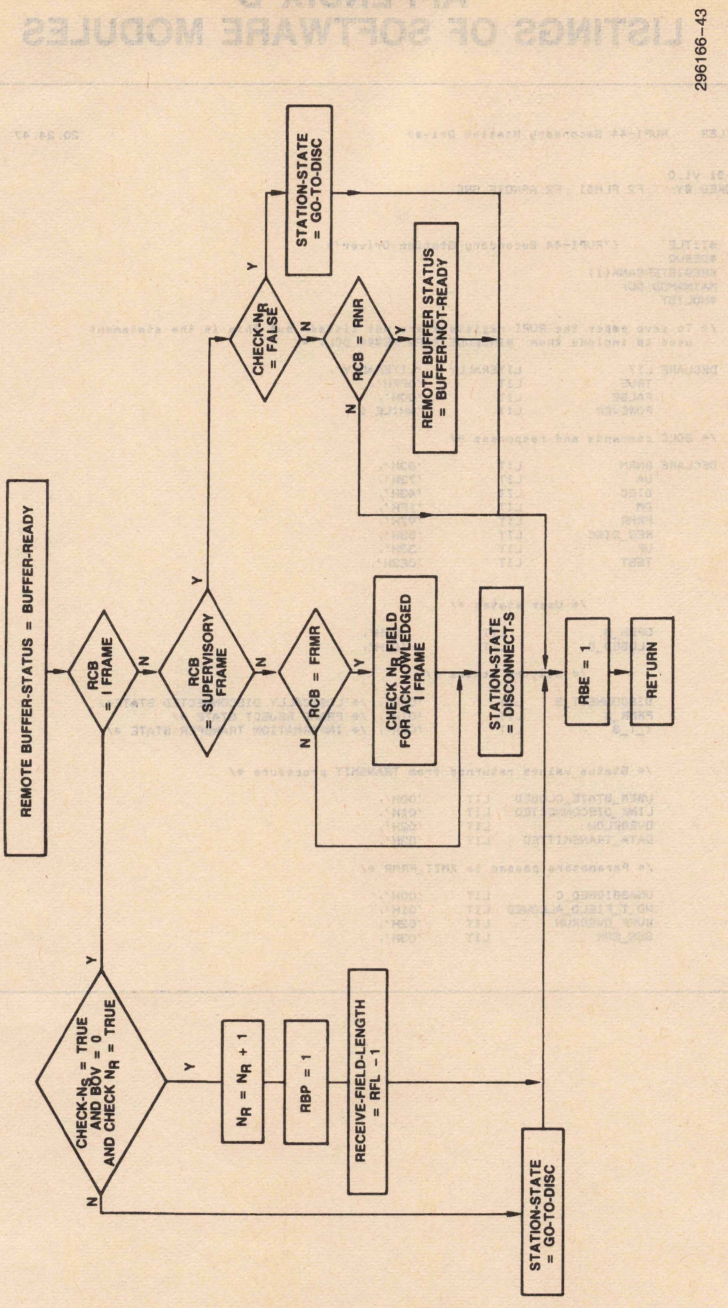


Figure 38. Primary Station Flow Charts

APPENDIX B
LISTINGS OF SOFTWARE MODULES

```
PL/M-51 COMPILER    RUPI-44 Secondary Station Driver                    20:24:47  09/20/83  PAGE   1

ISIS-II PL/M-51 V1.0
COMPILER INVOKED BY:   :F2:PLM51 :F2:APNOTE.SRC


                $TITLE      ('RUPI-44 Secondary Station Driver')
                $DEBUG
                $REGISTERBANK(1)
  1   1         MAIN$MOD:DO;
                $NOLIST

                /* To save paper the RUPI registers are not listed, but this is the statement
                   used to include them:  $INCLUDE (:F2:REG44.DCL) */

  5   1         DECLARE LIT             LITERALLY   'LITERALLY',
                        TRUE            LIT         '0FFH',
                        FALSE           LIT         '00H',
                        FOREVER         LIT         'WHILE 1';

                /* SDLC commands and responses */

  6   1         DECLARE SNRM            LIT         '83H',
                        UA              LIT         '73H',
                        DISC            LIT         '43H',
                        DM              LIT         '1FH',
                        FRMR            LIT         '97H',
                        REQ_DISC        LIT         '93H',
                        UP              LIT         '33H',
                        TEST            LIT         '0E3H',

                                /* User states */

                        OPEN_S          LIT         '00H',
                        CLOSED_S        LIT         '01H',

                                /* Station states */

                        DISCONNECT_S    LIT         '00H', /* LOGICALLY DISCONNECTED STATE*/
                        FRMR_S          LIT         '01H', /* FRAME REJECT STATE */
                        I_T_S           LIT         '02H', /* INFORMATION TRANSFER STATE */

                        /* Status values returned from TRANSMIT procedure */

                        USER_STATE_CLOSED   LIT     '00H',
                        LINK_DISCONNECTED   LIT     '01H',
                        OVERFLOW            LIT     '02H',
                        DATA_TRANSMITTED    LIT     '03H',

                        /* Parameters passed to XMIT_FRMR */

                        UNASSIGNED_C        LIT     '00H',
                        NO_I_FIELD_ALLOWED  LIT     '01H',
                        BUFF_OVERRUN        LIT     '02H',
                        SES_ERR             LIT     '03H',
```

296166-44

```
PL/M-51 COMPILER    RUPI-44 Secondary Station Driver                    20:24:47  09/20/83  PAGE   2


                        /* Variables */

                    USER_STATE          BYTE    AUXILIARY,
                    STATION_STATE       BYTE    AUXILIARY,
                    I_FRAME_LENGTH      BYTE    AUXILIARY,

                        /* Buffers */

                    BUFFER_LENGTH                       LIT     '60',
                    SIU_XMIT_BUFFER(BUFFER_LENGTH)      BYTE    PUBLIC    IDATA,
                    SIU_RECV_BUFFER(BUFFER_LENGTH)      BYTE    PUBLIC,
                    FRMR_BUFFER(3)              BYTE,

                        /* Flags */

                    XMIT_BUFFER_EMPTY       BIT PUBLIC;

  7   2     SIU_RECV: PROCEDURE (LENGTH) EXTERNAL;
  8   2         DECLARE LENGTH  BYTE;
  9   1     END SIU_RECV;

 10   2     OPEN: PROCEDURE PUBLIC USING 2;
 11   2             USER_STATE=OPEN_S;
 12   1     END OPEN;

 13   2     CLOSE: PROCEDURE PUBLIC USING 2;
 14   2             AM=0;
 15   2             USER_STATE=CLOSED_S;
 16   1     END CLOSE;

 17   2     POWER_ON_D: PROCEDURE   PUBLIC USING 0;

 18   2             USER_STATE=CLOSED_S;
 19   2             STATION_STATE=DISCONNECT_S;
 20   2             TBS=.SIU_XMIT_BUFFER(0);
 21   2             RBS=.SIU_RECV_BUFFER(0);
 22   2             RBL=BUFFER_LENGTH;
 23   2             RBE=1;      /* Enable the SIU's receiver */
 24   2             XMIT_BUFFER_EMPTY=1;

 25   1     END POWER_ON_D;

 26   2     TRANSMIT: PROCEDURE (XMIT_BUFFER_LENGTH) BYTE   PUBLIC USING 0;

            /* User must check XMIT_BUFFER_EMPTY flag before calling this procedure */

 27   2             DECLARE XMIT_BUFFER_LENGTH  BYTE,
                             I                  BYTE    AUXILIARY,
                             STATUS             BYTE    AUXILIARY;

 28   2             IF USER_STATE=CLOSED_S
                        THEN STATUS=USER_STATE_CLOSED;
 30   2             ELSE IF STATION_STATE=DISCONNECT_S
                             THEN STATUS=LINK_DISCONNECTED;
 32   2             ELSE IF XMIT_BUFFER_LENGTH>BUFFER_LENGTH
                             THEN STATUS=OVERFLOW;
 34   3             ELSE DO;
```

296166-45

```
PL/M-51 COMPILER    RUPI-44 Secondary Station Driver                    20:24:47  09/20/83  PAGE   3

   35   3                    XMIT_BUFFER_EMPTY=0;
   36   3                    TBL=XMIT_BUFFER_LENGTH;
   37   3                    I_FRAME_LENGTH=XMIT_BUFFER_LENGTH; /* Store length in case station
                                                                  is reset by FRMR, SNRM etc. */
   38   3                    TBF=1;
   39   3                    STATUS=DATA_TRANSMITTED;
   40   3                  END;
   41   2              RETURN STATUS;
   42   1      END TRANSMIT;

   43   2      XMIT_UNNUMBERED: PROCEDURE (CONTROL_BYTE) ;

   44   2          DECLARE CONTROL_BYTE    BYTE;

   45   2              TCB=CONTROL_BYTE;
   46   2              TBF=1;
   47   2              RTS=1;
   48   3              DO WHILE NOT SI;
   49   3              END;
   50   2              SI=0;

   51   1      END XMIT_UNNUMBERED;

   52   2      SNRM_RESPONSE: PROCEDURE ;

   53   2          STATION_STATE=I_T_S;
   54   2          NSNR=0;
   55   2          IF (RCB AND 10H) <> 0 /* Respond if polled */
                     THEN DO;
   57   3                    TBL=0;
   58   3                    CALL XMIT_UNNUMBERED(UA);
   59   3                END;
   60   2          IF XMIT_BUFFER_EMPTY=0  /* If an I frame was left pending transmission
                                             then restore it */
                   THEN DO;
   62   3                    TBL=I_FRAME_LENGTH;
   63   3                    TBF=1;
   64   3                END;
   65   2          AM=1;

   66   1      END SNRM_RESPONSE;


   67   2      XMIT_FRMR: PROCEDURE (REASON)  ;

   68   2              DECLARE REASON  BYTE;


   69   2              TCB=FRMR;

   70   2              TBS=.FRMR_BUFFER(0);
   71   2              TBL=3;
   72   2              FRMR_BUFFER(0)=RCB;
                       /* Swap nibbles in NSNR */
   73   2              FRMR_BUFFER(1)=(SHL((NSNR AND 0EH),4) OR  SHR((NSNR AND 0E0H),4));
   74   3              DO CASE REASON;
   75   3                  FRMR_BUFFER(2)=01H;   /* UNASSIGNED_C */
```

296166-46

```
PL/M-51 COMPILER    RUPI-44 Secondary Station Driver                20:24:47  09/20/83  PAGE   4

  76   3                 FRMR_BUFFER(2)=02H;   /* NO_I_FIELD_ALLOWED */
  77   3                 FRMR_BUFFER(2)=04H;   /* BUFF_OVERRUN */
  78   3                 FRMR_BUFFER(2)=08H;   /* SES_ERR */
  79   3              END;

  80   2            STATION_STATE=FRMR_S;

  81   2            IF (RCB AND 10H) <>0
                       THEN DO;
  83   3                         TBF=1;
  84   3                         RTS=1;
  85   4                         DO WHILE NOT SI;
  86   4                         END;
  87   3                         SI=0;
  88   3                       END;
  89   1      END XMIT_FRMR;


  90   2      IN_DISCONNECT_STATE: PROCEDURE ;/* Called from SIU_INT procedure */

  91   2              IF ((USER_STATE=OPEN_S) AND ((RCB AND 0EFH)=SNRM))
                         THEN CALL SNRM_RESPONSE;

  93   2              ELSE IF (RCB AND 10H) <> 0
                           THEN DO;
  95   3                             TBL=0;
  96   3                             CALL XMIT_UNNUMBERED(DM);
  97   3                           END;
  98   1      END IN_DISCONNECT_STATE;


  99   2      IN_FRMR_STATE: PROCEDURE ; /* Called by SIU_INT when a frame has been received
                                           when in the FRMR state */

 100   2          IF (RCB AND 0EFH)=SNRM
                   THEN DO;
 102   3                  CALL SNRM_RESPONSE;
 103   3                  TBS=.SIU_XMIT_BUFFER(0);  /* Restore transmit buffer start address */
 104   3                END;


 105   2          ELSE IF (RCB AND 0EFH)=DISC
                       THEN DO;
 107   3                      STATION_STATE=DISCONNECT_S;
 108   3                      TBS=.SIU_XMIT_BUFFER(0);   /* Restore transmit buffer start address */
 109   3                      IF (RCB AND 10H)<> 0
                                THEN DO;
 111   4                                TBL=0;
 112   4                                CALL XMIT_UNNUMBERED(UA);
 113   4                              END;
 114   3                    END;

 115   3          ELSE DO;       /* Receive control byte is something other than DISC or SNRM */
 116   3                 IF (RCB AND 10H) <> 0
                           THEN DO;
 118   4                             TBF=1;
 119   4                             RTS=1;
```

296166-47

```
PL/M-51 COMPILER    RUPI-44 Secondary Station Driver                    20:24:47  09/20/83  PAGE   5

 120   5                          DO WHILE NOT SI;
 121   5                          END;
 122   4                        END;
 123   3              END;

 124   1      END IN_FRMR_STATE;

 125   2      COMMAND_DECODE: PROCEDURE ;


 126   2          IF (RCB AND OEFH)=SNRM
                     THEN  CALL SNRM_RESPONSE;

 128   2          ELSE IF (RCB AND OEFH)=DISC
                         THEN  DO;
 130   3                          STATION_STATE=DISCONNECT_S;
 131   3                          IF (RCB AND 10H)<>0
                                     THEN DO;
 133   4                                  TBL=0;
 134   4                                  CALL XMIT_UNNUMBERED(UA);
 135   4                                END;
 136   3                        END;

 137   2          ELSE IF (RCB AND OEFH)=TEST
                         THEN DO;
 139   3                          IF (RCB AND 10H)>>0  /* Respond if polled */
                                     THEN DO;     /* FOR BOV=1, SEND THE TEST RESPONSE WITHOUT AN I FIELD */
 141   4                                  IF (BOV=1)
                                             THEN DO;
 143   5                                          TBL=0;
 144   5                                          CALL XMIT_UNNUMBERED(TEST OR 10H);
 145   5                                        END;
 146   5                                  ELSE DO; /* If no BOV, send received I field back to primary */
 147   5                                          TBL=RFL;
 148   5                                          TBS=RBS;
 149   5                                          CALL XMIT_UNNUMBERED(TEST OR 10H);
 150   5                                          TBS=.SIU_XMIT_BUFFER(0);  /* Restore TBS */
 151   5                                        END;

                                             /* If an I frame was pending, set it up again */

 152   4                                  IF XMIT_BUFFER_EMPTY=0
                                             THEN DO;
 154   5                                          TBL=I_FRAME_LENGTH;
 155   5                                          TBF=1;
 156   5                                        END;
 157   4                                END;
 158   3                          AM=1;
 159   3                        END;

 160   2          ELSE IF (RCB AND 01H) = 0 /* Kicked out of the AUTO mode because
                                               an I frame was received while RPB = 1 */
                         THEN DO;
 162   3                          AM = 1;
 163   3                          IF XMIT_BUFFER_EMPTY = 1
                                     THEN TBL = 0;
 165   3                          TBF = 1;    /* Send an AUTO mode response */
```

296166-48

```
PL/M-51 COMPILER    RUPI-44 Secondary Station Driver                    20:24:47  09/20/83  PAGE   6

  166   3                       RTS = 1;
  167   3                     END;

  168   2         ELSE CALL XMIT_FRMR(UNASSIGNED_C);  /* Received an undefined or not implemented command */

  169   1       END COMMAND_DECODE;


  170   2       SIU_INT: PROCEDURE INTERRUPT 4;

  171   2             DECLARE     I   BYTE    AUXILIARY;

  172   2             SI=0;
  173   2             IF STATION_STATE<> I_T_S /* Must be in NON-AUTO mode */
                        THEN DO;
  175   3                       IF RBE=0 /* Received a frame?  Give response */
                                  THEN DO;
  177   5                                 DO CASE STATION_STATE;
  178   5                                     CALL IN_DISCONNECT_STATE;
  179   5                                     CALL IN_FRMR_STATE;
  180   5                                 END;
  181   4                                 RBE=1;
  182   4                               END;
  183   3                     RETURN;
  184   3                     END;

                        /* If the program reaches this point, STATION_STATE=I_T_S
                            which means the SIU either was, or still is in the AUTO MODE */


  185   2             IF AM=0
                        THEN DO;
  187   3                       IF (RCB AND OEFH)=DISC
                                  THEN CALL COMMAND_DECODE;
  189   3                       ELSE IF USER_STATE=CLOSED_S
                                  THEN DO;
  191   4                                 TBL=0;
  192   4                                 CALL XMIT_UNNUMBERED(REQ_DISC);
  193   4                               END;
  194   3                       ELSE IF SES=1
                                  THEN CALL XMIT_FRMR(SES_ERR);
  196   3                       ELSE IF BOV=1
                                  THEN DO; /* DON'T SEND FRMR IF A TEST WAS RECEIVED*/
  198   4                                 IF (RCB AND OEFH)=TEST
                                            THEN CALL COMMAND_DECODE;
  200   4                                 ELSE     CALL XMIT_FRMR(BUFF_OVERRUN);
  201   4                               END;
  202   3                       ELSE CALL COMMAND_DECODE;
  203   3                       RBE=1;
  204   3                     END;

  205   3             ELSE DO; /* MUST STILL BE IN AUTO MODE */
  206   3                    IF TBF=0
                               THEN XMIT_BUFFER_EMPTY=1;  /* TRANSMITTED A FRAME */
  208   3                    IF RBE=0
                               THEN DO;
```

296166-49

```
PL/M-51 COMPILER    RUPI-44 Secondary Station Driver                    20:24:47  09/20/83  PAGE   7

  210   4                                 RBP=1; /* RNR STATE */
  211   4                                 RBE=1; /* RE-ENABLE RECEIVER */
  212   4                                 CALL SIU_RECV(RFL);
  213   4                                 RBP=0; /* RR STATE */
  214   4                               END;
  215   3                     END;
  216   1       END SIU_INT;

  217   1       END MAIN@MOD;

Software and application note written by Charles Yager


WARNINGS:
    4 IS THE HIGHEST USED INTERRUPT


MODULE INFORMATION:                 (STATIC+OVERLAYABLE)
    CODE SIZE                       = 028FH         655D
    CONSTANT SIZE                   = 0000H           0D
    DIRECT VARIABLE SIZE            =   3FH+02H     63D+  2D
    INDIRECT VARIABLE SIZE          =   3CH+00H     60D+  0D
    BIT SIZE                        =   01H+00H      1D+  0D
    BIT-ADDRESSABLE SIZE            =   00H+00H      0D+  0D
    AUXILIARY VARIABLE SIZE         = 0006H          6D
    MAXIMUM STACK SIZE              = 0017H         23D
    REGISTER-BANK(S) USED:            0 1 2
    460 LINES READ
    0 PROGRAM ERROR(S)
END OF PL/M-51 COMPILATION
```

296166-50

```
PL/M-51 COMPILER    Application Module: Async/SDLC Protocol converter          18:50:53  09/19/83  PAGE   1

ISIS-II PL/M-51 V1.0
COMPILER INVOKED BY:   :f2:plm51 :f2:unote.src


                $TITLE      ('Application Module: Async/SDLC Protocol converter')
                $debug
                $registerbank(0)
  1   1         user$mod:do;
                $NOLIST
  5   1         DECLARE     LIT                 LITERALLY       'LITERALLY',
                            TRUE                LIT             '0FFH',
                            FALSE               LIT             '00H',
                            FOREVER             LIT             'WHILE 1',
                            ESC                 LIT             '1BH',
                            LF                  LIT             '0AH',
                            CR                  LIT             '0DH',
                            BS                  LIT             '08H',
                            BEL                 LIT             '07H',
                            EMPTY               LIT             '00H',
                            INUSE               LIT             '01H',
                            FULL                LIT             '02H',
                            USER_STATE_CLOSED   LIT             '00H',
                            LINK_DISCONNECTED   LIT             '01H',
                            OVERFLOW            LIT             '02H',
                            DATA_TRANSMITTED    LIT             '03H',

                               /* BUFFERS */

                            BUFFER_LENGTH       LIT             '60',
                            SIU_XMIT_BUFFER(BUFFER_LENGTH)      BYTE    EXTERNAL    IDATA,
                            SIU_RECV_BUFFER(BUFFER_LENGTH)      BYTE    EXTERNAL,
                            FIFO_T(256)                 BYTE    AUXILIARY,
                            IN_PTR_T                    BYTE    AUXILIARY,
                            OUT_PTR_T                   BYTE    AUXILIARY,
                            BUFFER_STATUS_T             BYTE    AUXILIARY,
                            FIFO_R(256)                 BYTE    AUXILIARY,
                            IN_PTR_R                    BYTE    AUXILIARY,
                            OUT_PTR_R                   BYTE    AUXILIARY,
                            BUFFER_STATUS_R             BYTE    AUXILIARY,

                               /* Variables and Parameters */

                            LENGTH                      BYTE    AUXILIARY,
                            CHAR                        BYTE    AUXILIARY,
                            I                           BYTE    AUXILIARY,
                            USART_CMD                   BYTE    AUXILIARY,
                            DESTINATION_ADDRESS         BYTE    AUXILIARY,
                            SEND_DATA                   BYTE    AUXILIARY,
                            RESULT                      BYTE    AUXILIARY,
                            ERR_MESSAGE_INDEX           BYTE    AUXILIARY,
                            ERR_MESSAGE_PTR             WORD    AUXILIARY,

                              /*   Messages Sent to the Terminal   */

                            PARITY(*) BYTE CONSTANT(LF,CR,'Parity Error Detected',LF,CR,00H),
                            FRAME(*) BYTE CONSTANT(LF,CR,'Framing Error Detected',LF,CR,00H),
```

296166-51

```
PL/M-51 COMPILER    Application Module: Async/SDLC Protocol converter          18:50:53  09/19/83  PAGE   2

                    OVER_RUN(*) BYTE CONSTANT(LF,CR,'Overrun Error Detected',LF,CR,0),
                    LINK(*) BYTE CONSTANT(LF,CR,'Unable to Get Online',LF,CR,00H),
                    DEST_ADDR(*) BYTE CONSTANT(CR,LF,LF,
                              'Enter the destination address: __', BS, BS, 0),

                    D_ADDR_ACK(*) BYTE CONSTANT(CR,LF,LF,
                              'The new destination address is ',0),

                    STAT_ADDR(*) BYTE CONSTANT(CR,LF,LF,
                              'Enter the station address: __',BS,BS,0),

                    S_ADDR_ACK(*) BYTE CONSTANT(CR,LF,LF,
                              'The new station address is ',0),

                    ADDR_ACK_FIN(*) BYTE CONSTANT('H',CR,LF,LF,0),



                    SIGN_ON(*)  BYTE CONSTANT(CR, LF,LF,
                    '(\/)  RUPI-44 Secondary Station', CR, LF,
                    ' \/', CR, LF, LF,
                    '1 - Set the Station Address',LF,CR,
                    '2 - Set the Destination Address',CR,LF,
                    '3 - Go Online',CR,LF,
                    '4 - Go Offline',CR,LF,
                    '5 - Return to terminal mode',CR,LF,LF,
                    '    Enter option: _', BS, 0),

                    FIN(*)  BYTE CONSTANT(CR,LF,LF,0),


                    /* Characters Received From the Terminal */

                    HEX_TABLE(17) BYTE CONSTANT('0123456789ABCDEF',BEL),
                    MENU_CHAR(6) BYTE CONSTANT('12345',BEL),

                      /* Flags and Bits */

                    XMIT_BUFFER_EMPTY       BIT     EXTERNAL, /* Semaphore  for RUPI SIOU Transmit Buffer */
                    STOP_BIT                BIT     AT(147) REG, /* Terminal parameters */
                    ECHO                    BIT     AT(0B4H) REG,
                    WAIT                    BIT,            /* Timeout flag */
                    ERROR_FLAG              BIT,            /* Error message Flag */

                      /* Periheral Addresses */

                    USART_STATUS            BYTE    AT(0801H) AUXILIARY,
                    USART_DATA              BYTE    AT(0800H) AUXILIARY,
                    TIMER_CONTROL           BYTE    AT(1003H) AUXILIARY,
                    TIMER_0                 BYTE    AT(1000H) AUXILIARY,
                    TIMER_1                 BYTE    AT(1001H) AUXILIARY,
                    TIMER_2                 BYTE    AT(1002H) AUXILIARY;


                       /* External Procedures */
```

296166-52

```
PL/M-51 COMPILER    Application Module: Async/SDLC Protocol converter          18:50:53  09/19/83  PAGE   3


   6   2      POWER_ON_D: PROCEDURE   EXTERNAL;
   7   1      END POWER_ON_D;

   8   2      CLOSE: PROCEDURE EXTERNAL USING 2;
   9   1      END CLOSE;

  10   2      OPEN: PROCEDURE EXTERNAL USING 2;
  11   1      END OPEN;

  12   2      TRANSMIT:   PROCEDURE   (XMIT_BUFFER_LENGTH)    BYTE    EXTERNAL;
  13   2                  DECLARE XMIT_BUFFER_LENGTH  BYTE;
  14   1      END TRANSMIT;

                          /* Local Procedures */

  15   2      TIMER_0_INT: PROCEDURE INTERRUPT 1 USING 1;
  16   2          WAIT=0;
  17   1      END TIMER_0_INT;

  18   2      POWER_ON:   PROCEDURE USING 0;

  19   2              DECLARE TEMP    BYTE    AUXILIARY;

  20   2              SMD=54H;                /* Using DPLL, NRZI, PFS, TIMER 1, @ 62.5 Kbps */
  21   2              TMOD=21H;               /* Timer 0 16 bit, Timer 1 auto reload */
  22   2              TH1=0FFH;
  23   2              TCON=40H;

  24   2              TIMER_CONTROL=37H;    /* Initialize USART's system clock; 8254 */
  25   2              TIMER_0=04H;
  26   2              TIMER_0=00H;
  27   2              TIMER_CONTROL=77H;   /* Initialize TxC, RxC  */

              /*        Definition for dip switch tied to P1.0 to P1.6

                        Bit Rate    3       2       1
                       ------------------------------
                          300       on      on      on
                         1200       on      on      off
                         2400       on      off     on
                         4800       on      off     off
                         9600       off     on      on
                        19200       off     on      off

                        Stop bit     4
                       ------------------
                            1        on
                            2        off

                        Parity      6       5
                       ------------------------
                        off         on      on
                        odd         on      off
                        off         off     on
                        even        off     off
```

296166-53

```
PL/M-51 COMPILER    Application Module: Async/SDLC Protocol converter          18:50:53  09/19/83  PAGE   4


                        Echo        7
                        ____________
                        on          on
                        off         off                 */

  28   2              TEMP=P1 AND 07H;  /* Read the dip switch to determine the bit rate */
  29   2              IF TEMP>5
                         THEN TEMP=0;
  31   3              DO CASE TEMP;
                  /* 300 */
  32   4                  DO;
  33   4                      TIMER_1=83H;
  34   4                      TIMER_1=20H;
  35   4                  END;

  36   4              /* 1200 */  DO;
  37   4                      TIMER_1=20H;
  38   4                      TIMER_1=05H;
  39   4                  END;

  40   4              /* 2400 */  DO;
  41   4                      TIMER_1=60H;
  42   4                      TIMER_1=02H;
  43   4                  END;

  44   4              /* 4800 */  DO;
  45   4                      TIMER_1=30H;
  46   4                      TIMER_1=01H;
  47   4                  END;

  48   4              /* 9600 */  DO;
  49   4                      TIMER_1=65H;
  50   4                      TIMER_1=0;
  51   4                  END;

  52   4              /* 19200 */ DO;
  53   4                      TIMER_1=33H;
  54   4                      TIMER_1=0;
  55   4                  END;
  56   3              END;

  57   2              USART_STATUS=0;   /* Software power-on reset for 8251A */
  58   2              USART_STATUS=0;
  59   2              USART_STATUS=0;
  60   2              USART_STATUS=40H;

  61   2              TEMP=0AH;         /* Determine the parity and # of stop bits */
  62   2              TEMP=TEMP OR (P1 AND 30H);
  63   2              IF STOP_BIT=1
                         THEN TEMP=TEMP OR 0C0H;
  65   2              ELSE TEMP=TEMP OR 40H;

  66   2              USART_STATUS=TEMP;   /* USART Mode Word */
  67   2              USART_STATUS, USART_CMD=27H; /*USART Command Word RTS, RxE, DTR, TxEN=1*/

  68   2              STAD=0FFH;
```

296166-54

```
PL/M-51 COMPILER    Application Module: Async/SDLC Protocol converter          18:50:53  09/19/83  PAGE   5


  69   2              SEND_DATA=0; /* Intialize Flags */

  70   2             IN_PTR_T, OUT_PTR_T, IN_PTR_R, OUT_PTR_R = 0;/*Initialize FIFO PTRs*/

  71   2              BUFFER_STATUS_T, BUFFER_STATUS_R= EMPTY;

  72   2              CALL POWER_ON_D;

  73   2              IP=01H;           /* USART's RxRdy is the highest priority */
                                        /* Both external interrupts are level triggered*/
  74   2              IE=93H;           /* Enable USART RxRdy, SI, and Timer 0 interrupts*/

  75   2              ERROR_FLAG=0;

  76   1      END POWER_ON;

  77   2      FIFO_R_IN: PROCEDURE (CHAR) USING 1;
  78   2          DECLARE CHAR    BYTE;

  79   2          FIFO_R(IN_PTR_R)=CHAR;
  80   2          IN_PTR_R=IN_PTR_R+1;

  81   2          IF BUFFER_STATUS_R=EMPTY
                  THEN DO;
  83   3                  EA=0;
  84   3                  BUFFER_STATUS_R=INUSE;
  85   3                  EX1=1;        /* Enable USART's TxD interrupt */
  86   3                  EA=1;
  87   3                END;
  88   2          ELSE IF ((BUFFER_STATUS_R=INUSE) AND (IN_PTR_R=OUT_PTR_R))
                      THEN BUFFER_STATUS_R=FULL;

  90   1      END FIFO_R_IN;

  91   2      FIFO_R_OUT: PROCEDURE BYTE USING 1;

  92   2          DECLARE CHAR    BYTE    AUXILIARY;

  93   2          CHAR=FIFO_R(OUT_PTR_R);
  94   2          OUT_PTR_R=OUT_PTR_R+1;
  95   2          IF OUT_PTR_R=IN_PTR_R
                  THEN DO;
  97   3                  EX1=0;  /* Shut off TxD interrupt */
  98   3                  BUFFER_STATUS_R=EMPTY;
  99   3                END;
 100   2          ELSE IF ((BUFFER_STATUS_R=FULL) AND (OUT_PTR_R-20=IN_PTR_R))
                      THEN BUFFER_STATUS_R=INUSE;

 102   2          RETURN CHAR;

 103   1      END FIFO_R_OUT;


 104   2      USART_XMIT_INT: PROCEDURE INTERRUPT 2 USING 1;
```

296166-55

```
PL/M-51 COMPILER    Application Module: Async/SDLC Protocol converter                18:50:53  09/19/83  PAGE   6

 105   2          DECLARE
                        MESSAGE BASED ERR_MESSAGE_PTR(1)    BYTE    CONSTANT;

 106   2          IF ERROR_FLAG
                    THEN DO;
 108   3                  IF MESSAGE(ERR_MESSAGE_INDEX)<>0  /* Then continue to send the message */
                            THEN DO;
 110   4                          USART_DATA = MESSAGE(ERR_MESSAGE_INDEX);
 111   4                          ERR_MESSAGE_INDEX=ERR_MESSAGE_INDEX+1;
 112   4                        END;

 113   4                  ELSE DO; /* If message is done reset ERROR_FLAG and shut off interrupt if FIFO is empty */
 114   4                         ERROR_FLAG=0;
 115   4                         IF BUFFER_STATUS_R = EMPTY
                                   THEN EX1=0;
 117   4                       END;
 118   3                END;

 119   2          ELSE USART_DATA=FIFO_R_OUT;

 120   1      END USART_XMIT_INT;


 121   2      SIU_RECV: PROCEDURE (LENGTH) PUBLIC USING 1;

 122   2          DECLARE LENGTH  BYTE,
                         I       BYTE    AUXILIARY;

 123   3          DO I=0 TO LENGTH-1;
 124   4              DO WHILE BUFFER_STATUS_R=FULL;/* Check to see if fifo is full */
 125   4              END;
 126   3              CALL FIFO_R_IN(SIU_RECV_BUFFER(I));
 127   3          END;

 128   1      END SIU_RECV;


 129   2      FIFO_T_IN: PROCEDURE (CHAR) USING 2;

 130   2          DECLARE CHAR        BYTE;

 131   2          FIFO_T(IN_PTR_T)=CHAR;
 132   2          IN_PTR_T=IN_PTR_T+1;
 133   2          IF CHAR=LF
                    THEN SEND_DATA=SEND_DATA+1;

 135   2          IF BUFFER_STATUS_T=EMPTY
                    THEN BUFFER_STATUS_T=INUSE;
 137   2          ELSE IF ((BUFFER_STATUS_T=INUSE) AND (IN_PTR_T+20=OUT_PTR_T))
                         THEN DO; /* Stop reception using CTS */
 139   3                         USART_STATUS, USART_CMD=USART_CMD AND NOT(20H);
 140   3                         BUFFER_STATUS_T=FULL;
 141   3                         IF SEND_DATA=0
                                   THEN SEND_DATA=1;/*If the buffer is full and no LF
                                                      has been received then send data */
 143   3                       END;
 144   1      END FIFO_T_IN;
```

296166-56

```
PL/M-51 COMPILER    Application Module: Async/SDLC Protocol converter                    18:30:53  09/19/83  PAGE   7


  145   2       FIFO_T_OUT: PROCEDURE BYTE ;

  146   2           DECLARE CHAR     BYTE     AUXILIARY;

  147   2           CHAR=FIFO_T(OUT_PTR_T);
  148   2           OUT_PTR_T=OUT_PTR_T+1;
  149   2           IF OUT_PTR_T=IN_PTR_T    /* Then FIFO_T is empty */
                      THEN DO;
  151   3                    EA=0;
  152   3                    BUFFER_STATUS_T=EMPTY;
  153   3                    SEND_DATA=0;
  154   3                    EA=1;
  155   3                END;
  156   2           ELSE IF ((BUFFER_STATUS_T=FULL) AND (OUT_PTR_T-80=IN_PTR_T))
                           THEN DO;
  158   3                             USART_STATUS, USART_CMD=USART_CMD OR 20H;
  159   3                             BUFFER_STATUS_T=INUSE;
  160   3                        END;
  161   2           IF (CHAR=LF AND SEND_DATA>0) THEN SEND_DATA=SEND_DATA-1;
  163   2           RETURN CHAR;
  164   1       END FIFO_T_OUT;

  165   2       ERROR: PROCEDURE (STATUS) USING 2;

  166   2               DECLARE STATUS       BYTE;

  167   2               IF (STATUS AND 08H)<>0
                          THEN ERR_MESSAGE_PTR=.PARITY;
  169   2               ELSE IF (STATUS AND 10H)<>0
                          THEN ERR_MESSAGE_PTR=.OVER_RUN;
  171   2               ELSE IF (STATUS AND 20H)<>0
                          THEN ERR_MESSAGE_PTR=.FRAME;

  173   2               USART_STATUS=(USART_CMD OR 10H); /* Reset error flags on USART */

  174   2               ERR_MESSAGE_INDEX = 0;
  175   2               ERROR_FLAG=1;
  176   2               EX1=1;              /* Turn on Tx Interrupt */

  177   1       END ERROR;

  178   2       LINK_DISC: PROCEDURE ;

                   /* This procedure sends the message 'Unable to Get Online' to the terminal */

  179   2               DECLARE MESSAGE_PTR WORD     AUXILIARY,
                               MESSAGE     BASED    MESSAGE_PTR(1)      BYTE    CONSTANT,
                               J           BYTE     AUXILIARY,
                               EX1_STORE   BIT;


  180   2               EX1_STORE=EX1;     /* Shut off async transmit interrupt */
  181   2               EX1=0;
  182   2               MESSAGE_PTR=.LINK;
  183   2               J=0;
  184   3               DO WHILE (MESSAGE(J)<>0);
```

296166-57

```
PL/M-51 COMPILER    Application Module: Async/SDLC Protocol converter    18:50:53  09/19/83  PAGE   8

  185   4              DO WHILE (USART_STATUS AND 01H)=0;   /* Wait for TxRDY on USART */
  186   4              END;
  187   3              USART_DATA=MESSAGE(J);
  188   3              J=J+1;
  189   3          END;
  190   2          EX1=EX1_STORE;    /* Restore async transmit interrupt */
  191   1      END LINK_DISC;

  192   2      CO: PROCEDURE (CHAR) USING 2;
  193   2          DECLARE CHAR    BYTE;

  194   3          DO WHILE (USART_STATUS AND 01H) = 0;
  195   3          END;
  196   2          USART_DATA=CHAR;

  197   1      END CO;

  198   2      CI: PROCEDURE BYTE USING 2;

  199   3          DO WHILE (USART_STATUS AND 02H) = 0;
  200   3          END;
  201   2          RETURN USART_DATA;

  202   1      END CI;

  203   2      GET_HEX: PROCEDURE BYTE USING 2;

  204   2          DECLARE CHAR    BYTE    AUXILIARY,
                          I       BYTE    AUXILIARY;

  205   2          LO: CHAR=CI;

  206   3              DO I=0 TO 15;
  207   3                  IF CHAR=HEX_TABLE(I)
                              THEN GOTO L1;
  209   3              END;

  210   2          L1: CALL CO(HEX_TABLE(I));
  211   2              IF I=16
                          THEN GOTO LO;

  213   2              RETURN I;

  214   1      END GET_HEX;

  215   2      OUTPUT_MESSAGE: PROCEDURE (MESSAGE_PTR) USING 2;
  216   2          DECLARE MESSAGE_PTR WORD,
                          MESSAGE     BASED   MESSAGE_PTR(1)  BYTE CONSTANT,
                          I           BYTE    AUXILIARY;

  217   2          I=0;

  218   3          DO WHILE MESSAGE(I) <> 0;
  219   3              CALL CO(MESSAGE(I));
  220   3              I=I+1;
```

296166-58

```
PL/M-51 COMPILER    Application Module: Async/SDLC Protocol converter          18:50:53  09/19/83  PAGE   9

221   3          END;

222   1      END OUTPUT_MESSAGE;


223   2      MENU: PROCEDURE USING 2;

224   2          DECLARE I               BYTE    AUXILIARY,
                      CHAR              BYTE    AUXILIARY,
                      STATION_ADDRESS   BYTE    AUXILIARY;


225   2      START:
               CALL OUTPUT_MESSAGE(.SIGN_ON);

226   2          MO: CHAR=CI;    /* Read a character */

227   3              DO I=0 TO 4;
228   3                  IF CHAR=MENU_CHAR(I)
                            THEN GOTO M1;
230   3              END;

231   2          M1: CALL CO(MENU_CHAR(I));
232   2              IF I=5
                        THEN GOTO MO;

234   3          DO CASE I;

235   4              DO;
236   4                  CALL OUTPUT_MESSAGE(.STAT_ADDR);

237   4                  STATION_ADDRESS=SHL(GET_HEX,4);

238   4                  STATION_ADDRESS=(STATION_ADDRESS OR GET_HEX);

239   4                  STAD=STATION_ADDRESS;

240   4                  CALL OUTPUT_MESSAGE(.S_ADDR_ACK);

241   4                  CALL CO(HEX_TABLE(SHR(STATION_ADDRESS,4)));
242   4                  CALL CO(HEX_TABLE(OFH AND STATION_ADDRESS));

243   4                  CALL OUTPUT_MESSAGE(.ADDR_ACK_FIN);
244   4              END;

245   4              DO;

246   4                  CALL OUTPUT_MESSAGE(.DEST_ADDR);

247   4                  DESTINATION_ADDRESS=SHL(GET_HEX,4);

248   4                  DESTINATION_ADDRESS=(DESTINATION_ADDRESS OR GET_HEX );

249   4                  CALL OUTPUT_MESSAGE(.D_ADDR_ACK);
```

296166-59

```
PL/M-51 COMPILER    Application Module: Async/SDLC Protocol converter                18:50:53  09/19/83  PAGE  10

  250   4                   CALL CO(HEX_TABLE(SHR(DESTINATION_ADDRESS,4)));
  251   4                   CALL CO(HEX_TABLE(OFH AND DESTINATION_ADDRESS));

  252   4                   CALL OUTPUT_MESSAGE(.ADDR_ACK_FIN);
  253   4               END;

  254   4               DO;
  255   4                   CALL OUTPUT_MESSAGE(.FIN);
  256   4                   CALL OPEN;
  257   4               END;


  258   4               DO;
  259   4                   CALL OUTPUT_MESSAGE(.FIN);
  260   4                   CALL CLOSE;
  261   4               END;

  262   3               CALL OUTPUT_MESSAGE(.FIN);

  263   3           END; /* DO CASE */

  264   1       END MENU;


  265   2       USART_RECV_INT: PROCEDURE INTERRUPT 0 USING 2;

  266   2           DECLARE CHAR         BYTE    AUXILIARY,
                       STATUS       BYTE    AUXILIARY;


  267   2           CHAR=USART_DATA;
  268   2           STATUS=USART_STATUS AND 38H;
  269   2           IF STATUS<>0
                      THEN CALL ERROR(STATUS);
  271   2           ELSE IF CHAR=ESC
                           THEN CALL MENU;
  273   3           ELSE DO;
  274   3                   CALL FIFO_T_IN(CHAR);
  275   3                   IF ECHO=0
                               THEN CALL CO(CHAR);
  277   3                END;

  278   1       END USART_RECV_INT;


  279   1       BEGIN:
               CALL POWER_ON;

  280   2           DO FOREVER;
  281   2               IF SEND_DATA>0
                           THEN DO;
  283   4                               DO WHILE NOT(XMIT_BUFFER_EMPTY); /*Wait until SIU_XMIT_BUFFer
                                                                          is empty */
  284   4                               END;
  285   3                               LENGTH, CHAR =1;
  286   3                               SIU_XMIT_BUFFER(0)=DESTINATION_ADDRESS;
  287   4                               DO WHILE ((CHAR<>LF) AND (LENGTH<BUFFER_LENGTH) AND (BUFFER_STATUS_T<>EMPTY));
```

296166-60

```
PL/M-51 COMPILER    Application Module: Async/SDLC Protocol converter                18:50:53  09/19/83  PAGE  11

  288   4                           CHAR=FIFO_T_OUT;
  289   4                           SIU_XMIT_BUFFER(LENGTH)=CHAR;
  290   4                           LENGTH=LENGTH+1;
  291   4                       END;

            /* If the line entered at the terminal is greater than BUFFER_LENGTH char, send the
               first BUFFER_LENGTH char, then send the rest; since the SIU buffer is only BUFFER_LENGTH bytes */
  292   3         L1:           I=0; /* Use I to count the number of unsuccesful
                                    transmits */

  293   3         RETRY:      RESULT=TRANSMIT(LENGTH); /* Send the message */
  294   3                       IF RESULT<>DATA_TRANSMITTED
                                  THEN DO;
                              /* Wait 50 msec for link to connect then try again */
  296   4                           WAIT=1;
  297   4                           TH0=3CH;
  298   4                           TL0=0AFH;
  299   4                           TR0=1;
  300   5                           DO WHILE WAIT;
  301   5                           END;
  302   4                           TR0=0;
  303   4                           I=I+1;
  304   5                           IF I>100 THEN DO;  /* Wait 5 sec to get on line else
                                                         send error message to terminal
                                                         and try again  */
  306   5                                                CALL LINK_DISC;
  307   5                                                GOTO L1;
  308   5                                              END;
  309   4                           GOTO RETRY;
  310   4                         END;
  311   3                   END;

  312   2           END;
  313   1         END USER$MOD;



WARNINGS:
    2 IS THE HIGHEST USED INTERRUPT


MODULE INFORMATION:                 (STATIC+OVERLAYABLE)
    CODE SIZE                       = 06B2H       1714D
    CONSTANT SIZE                   = 01CFH        463D
    DIRECT VARIABLE SIZE            =   00H+05H      0D+  5D
    INDIRECT VARIABLE SIZE          =   00H+00H      0D+  0D
    BIT SIZE                        =   02H+01H      2D+  1D
    BIT-ADDRESSABLE SIZE            =   00H+00H      0D+  0D
    AUXILIARY VARIABLE SIZE         = 021FH        543D
    MAXIMUM STACK SIZE              = 002BH         40D
    REGISTER-BANK(S) USED:            0 1 2
    713 LINES READ
    0 PROGRAM ERROR(S)
END OF PL/M-51 COMPILATION
```

296166-61

```
PL/M-51 COMPILER    RUPI-44 Primary Station                    20:47:13  09/26/83  PAGE   1

ISIS-II PL/M-51 V1.0
COMPILER INVOKED BY:  :F2:PLM51 :F2:PNOTE.SRC


                $TITLE      ('RUPI-44 Primary Station')
                $DEBUG
                $REGISTERBANK(0)
   1    1       MAIN$MOD:DO;

                /* To save paper the RUPI registers are not listed, but this is the statement
                   used to include them:  $INCLUDE (:F2:REG44.DCL) */

                $NOLIST

   5    1       DECLARE LIT             LITERALLY   'LITERALLY',
                        TRUE            LIT         '0FFH',
                        FALSE           LIT         '00H',
                        FOREVER         LIT         'WHILE 1';

                /* SDLC COMMANDS AND RESPONSES */

   6    1       DECLARE SNRM            LIT         '93H',
                        UA              LIT         '73H',
                        DISC            LIT         '53H',
                        DM              LIT         '1FH',
                        FRMR            LIT         '97H',
                        REQ_DISC        LIT         '53H',
                        UP              LIT         '33H',
                        TEST            LIT         '0F3H',
                        RR              LIT         '11H',
                        RNR             LIT         '15H',


                              /* REMOTE STATION BUFFER STATUS */

                        BUFFER_READY        LIT         '0',
                        BUFFER_NOT_READY    LIT         '1',

                              /* STATION STATES */
                        DISCONNECT_S    LIT         '00H', /* LOGICALLY DISCONNECTED STATE*/
                        GO_TO_DISC      LIT         '01H',
                        I_T_S           LIT         '02H', /* INFORMATION TRANSFER STATE */


                        /* PARAMETERS PASSED TO XMIT_I_T_S */
                        T_I_FRAME           LIT     '00H',
                        T_RR                LIT     '01H',
                        T_RNR               LIT     '02H',


                        /* SECONDARY STATION IDENTIFICATION */

                        NUMBER_OF_STATIONS  LIT     '2',
                        SECONDARY_ADDRESSES(NUMBER_OF_STATIONS)
                                            BYTE    CONSTANT(55H,43H),
```

296166-62

```
PL/M-51 COMPILER    RUPI-44 Primary Station                    20:47:13  09/26/83  PAGE   2


                        /* Remote Station Database */

                    RSD(NUMBER_OF_STATIONS) STRUCTURE
                    (STATION_ADDRESS     BYTE,
                     STATION_STATE       BYTE,
                     NS                  BYTE,
                     NR                  BYTE,
                     BUFFER_STATUS       BYTE,  /* The status of the secondary stations buffer */
                     INFO_LENGTH         BYTE,
                     DATA(64)            BYTE)   AUXILIARY,

                        /* VARIABLES */
                    STATION_NUMBER       BYTE    AUXILIARY,
                    RECV_FIELD_LENGTH    BYTE    AUXILIARY,
                    WAIT                 BIT,

                        /* BUFFERS */
                    SIU_XMIT_BUFFER(64)      BYTE    IDATA,
                    SIU_RECV_BUFFER(64)      BYTE;


  7   2     POWER_ON: PROCEDURE ;

  8   2         DECLARE I   BYTE    AUXILIARY;

  9   2             TBS=.SIU_XMIT_BUFFER(0);
 10   2             RBS=.SIU_RECV_BUFFER(0);
 11   2             RBL=64;       /* 64 Byte receive buffer */
 12   2             RBE=1;        /* Enable the SIU's receiver */

 13   3             DO I= 0 TO NUMBER_OF_STATIONS-1;

 14   3                 RSD(I).STATION_ADDRESS=SECONDARY_ADDRESSES(I);
 15   3                 RSD(I).STATION_STATE=DISCONNECT_S;
 16   3                 RSD(I).BUFFER_STATUS=BUFFER_NOT_READY;
 17   3                 RSD(I).INFO_LENGTH=0;

 18   3             END;

 19   2             SMD=54H; /* Using DPLL, NRZI, PFS, TIMER 1, @ 62.5 Kbps */
 20   2             TMOD=21H;
 21   2             TH1=0FFH;
 22   2             TCON=40H; /* Use timer 0 for receive time out interrupt */
 23   2             IE=82H;

 24   1     END POWER_ON;

 25   2     XMIT: PROCEDURE (CONTROL_BYTE);

 26   2         DECLARE CONTROL_BYTE    BYTE;

 27   2             TCB=CONTROL_BYTE;
 28   2             TBF=1;
```

296166-63

```
PL/M-51 COMPILER    RUPI-44 Primary Station                    20:47:13  09/26/83  PAGE   3

  29   2              RTS=1;
  30   3              DO WHILE NOT SI;
  31   3              END;
  32   2              SI=0;

  33   1      END XMIT;


  34   2      TIMER_0_INT: PROCEDURE INTERRUPT 1  USING 1;
  35   2          WAIT=0;
  36   1      END TIMER_0_INT;


  37   2      TIME_OUT: PROCEDURE BYTE;          /* Time_out returns true if there wasn't
                                                    a frame received within 200 msec.
                                                    If there was a frame received within
                                                    200 msec then time_out returns false. */
  38   2          DECLARE     I   BYTE    AUXILIARY;

  39   3          DO I=0 TO 3;

  40   3              WAIT=1;
  41   3              TH0=3CH;
  42   3              TL0=0AFH;
  43   3              TR0=1;
  44   4                  DO WHILE WAIT;
  45   4                      IF SI=1
                                 THEN GOTO T_01;
  47   4                  END;
  48   3          END;

  49   2          RETURN TRUE;

  50   2      T_01:
                  SI=0;
  51   2              RETURN FALSE;

  52   1      END TIME_OUT;

  53   2      SEND_DISC: PROCEDURE;

  54   2                  TBL=0;
  55   2                  CALL XMIT(DISC);
  56   2                  IF TIME_OUT=FALSE
                              THEN IF RCB=UA OR RCB=DM
  57   3                                 THEN DO;
  59   3                                      RSD(STATION_NUMBER).BUFFER_STATUS=BUFFER_NOT_READY;
  60   3                                      RSD(STATION_NUMBER).STATION_STATE=DISCONNECT_S;
  61   3                                  END;
  62   2                  RBE=1;

  63   1      END SEND_DISC;


  64   2      SEND_SNRM: PROCEDURE;
  65   2          TBL=0;
```

296166-64

```
PL/M-51 COMPILER    RUPI-44 Primary Station                    20:47:13  09/26/83  PAGE   4

  66   2          CALL XMIT(SNRM);
  67   2          IF (TIME_OUT=FALSE) AND (RCB=UA)
                        THEN DO;
  69   3                          RSD(STATION_NUMBER).STATION_STATE=I_T_S;
  70   3                          RSD(STATION_NUMBER).NS=0;
  71   3                          RSD(STATION_NUMBER).NR=0;
  72   3                        END;
  73   2          RBE=1;

  74   1      END SEND_SNRM;


  75   2      CHECK_NS: PROCEDURE BYTE;

                /* Check the Ns Field of the received frame.  If Nr(P)=Ns(S) return true */

  76   2          IF (RSD(STATION_NUMBER).NR=(SHR(RCB,1) AND 07H))
                      THEN RETURN TRUE;
  78   2                  ELSE RETURN FALSE;

  79   1      END CHECK_NS;

  80   2      CHECK_NR: PROCEDURE BYTE;

              /* Check the Nr field of the received frame.  If Ns(P)+1=Nr(S) then the frame
                  has been acknowledged, else if Ns(P)=Nr(S) then the frame has not been
                  acknowledged, else reset the secondary */

  81   2          IF (((RSD(STATION_NUMBER).NS + 1) AND 07H) = SHR(RCB,5))
                      THEN DO;
  83   3                      RSD(STATION_NUMBER).NS=((RSD(STATION_NUMBER).NS+1) AND 07H);
  84   3                      RSD(STATION_NUMBER).INFO_LENGTH=0;
  85   3                    END;
  86   2          ELSE IF (RSD(STATION_NUMBER).NS <> SHR(RCB,5))
                         THEN RETURN FALSE;

  88   2          RETURN TRUE;

  89   1      END CHECK_NR;


  90   2      RECEIVE: PROCEDURE ;

  91   2            DECLARE     I   BYTE    AUXILIARY;

  92   2            RSD(STATION_NUMBER).BUFFER_STATUS=BUFFER_READY;

               /* If an RNR was received buffer_status will be changed in the supervisory
                  frame decode section futher down in this procedure, any other response
                  means the remote stations buffer is ready  */

  93   2            IF (RCB AND 01H)=0
                      THEN DO;  /* I Frame Received */
  95   3                      IF (CHECK_NS=TRUE AND BOV=0 AND CHECK_NR=TRUE)
                                 THEN DO;
  97   4                                 RSD(STATION_NUMBER).NR=((RSD(STATION_NUMBER).NR+1) AND 07H);
  98   4                                 RBP=1;
```

296166-65

```
PL/M-51 COMPILER    RUPI-44 Primary Station                          20:47:13  09/26/83  PAGE   5

 99   4                                    RECV_FIELD_LENGTH=RFL-1;
100   4                                END;

101   3                          ELSE RSD(STATION_NUMBER).STATION_STATE=GO_TO_DISC;
102   3                       END;
103   2                ELSE IF (RCB AND 03H)=01H
                          THEN DO;  /* Supervisory frame received */
105   3                               IF CHECK_NR=FALSE
                                        THEN RSD(STATION_NUMBER).STATION_STATE=GO_TO_DISC;

107   3                               ELSE IF ((RCB AND 0FH)=05H) /* then RNR */
                                           THEN RSD(STATION_NUMBER).BUFFER_STATUS=BUFFER_NOT_READY;
109   3                             END;

110   3                ELSE DO;          /* Unnumbered frame or unknown frame received */
111   3                        IF RCB=FRMR
                               THEN DO;    /* If FRMR was received check Nr for an
                                              acknowledged I frame  */
113   4                                  RCB=SIU_RECV_BUFFER(1);
114   4                                  I=CHECK_NR;
115   4                               END;
116   3                        RSD(STATION_NUMBER).STATION_STATE=GO_TO_DISC;
117   3                    END;

118   2                RBE=1;

119   1      END RECEIVE;



120   2      XMIT_I_T_S: PROCEDURE (TEMP);

121   2          DECLARE     TEMP    BYTE;

122   2              IF TEMP=T_I_FRAME
                       THEN DO;    /* Transmit I frame */
                                   /* Transfer the station buffer into internal ram */

124   4                              DO TEMP=0 TO RSD(STATION_NUMBER).INFO_LENGTH-1;
125   4                              SIU_XMIT_BUFFER(TEMP)=RSD(STATION_NUMBER).DATA(TEMP);
126   4                              END;

                                     /* Build the I frame control field */

127   3                              TEMP=(SHL(RSD(STATION_NUMBER).NR,5) OR SHL(RSD(STATION_NUMBER).NS,1) OR 10H);
128   3                              TBL=RSD(STATION_NUMBER).INFO_LENGTH;
129   3                              CALL XMIT(TEMP);
130   3                              IF TIME_OUT=FALSE
                                        THEN CALL RECEIVE;
132   3                         END;

133   3                ELSE DO;    /* Transmit RR or RNR*/
134   3                        IF TEMP=T_RR
                                    THEN TEMP=RR;
136   3                        ELSE TEMP=RNR;
```

296166-66

```
PL/M-51 COMPILER    RUPI-44 Primary Station                    20:47:13  09/26/83  PAGE   6

 137   3                    TEMP=(SHL(RSD(STATION_NUMBER).NR,5) OR TEMP);
 138   3                    TBL=0;
 139   3                    CALL XMIT(TEMP);
 140   3                    IF TIME_OUT=FALSE
                                THEN CALL RECEIVE;
 142   3                END;
 143   1       END XMIT_I_T_S;

 144   2       BUFFER_TRANSFER: PROCEDURE;

 145   2           DECLARE     I       BYTE    AUXILIARY,
                               J       BYTE    AUXILIARY;


 146   3           DO I=0 TO NUMBER_OF_STATIONS-1;
 147   3               IF RSD(I).STATION_ADDRESS=SIU_RECV_BUFFER(0)
                           THEN GOTO T1;
 149   3           END;

 150   2         T1:   IF I=NUMBER_OF_STATIONS  /* If the addressed station does not exits,
                                                   then discard the data */
                           THEN DO;
 152   3                          RBP=0;
 153   3                          RETURN;
 154   3                      END;
 155   2                ELSE IF RSD(I).INFO_LENGTH=0
                           THEN DO;
 157   3                          RSD(I).INFO_LENGTH=RECV_FIELD_LENGTH;
 158   4                          DO J=1 TO RECV_FIELD_LENGTH;
 159   4                              RSD(I).DATA(J-1)=SIU_RECV_BUFFER(J);
 160   4                          END;
 161   3                          RBP=0;
 162   3                      END;

 163   1       END BUFFER_TRANSFER;


 164   1       BEGIN:
               CALL POWER_ON;

 165   2           DO FOREVER;

 166   3               DO STATION_NUMBER=0 TO NUMBER_OF_STATIONS-1;
 167   3                   STAD=RSD(STATION_NUMBER).STATION_ADDRESS;
 168   3                   IF RSD(STATION_NUMBER).STATION_STATE = DISCONNECT_S
                               THEN CALL SEND_SNRM;
 170   3                   ELSE IF RSD(STATION_NUMBER).STATION_STATE = GO_TO_DISC
                               THEN CALL SEND_DISC;
 172   3                   ELSE IF ((RSD(STATION_NUMBER).INFO_LENGTH>0) AND
                                    (RSD(STATION_NUMBER).BUFFER_STATUS=BUFFER_READY))
                               THEN CALL XMIT_I_T_S(T_I_FRAME);
 174   3                   ELSE IF RBP=0
                               THEN CALL XMIT_I_T_S(T_RR);
 176   3                   ELSE CALL XMIT_I_T_S(T_RNR);

 177   3                   IF RBP=1
                               THEN CALL BUFFER_TRANSFER;
```

296166-67

```
PL/M-51 COMPILER    RUPI-44 Primary Station                    20:47:13  09/26/83  PAGE   7

 179   3               END;

 180   2           END;

 181   1       END MAIN$MOD;



WARNINGS:
    1 IS THE HIGHEST USED INTERRUPT


MODULE INFORMATION:                 (STATIC+OVERLAYABLE)
    CODE SIZE                       = 053DH       1341D
    CONSTANT SIZE                   = 0002H         2D
    DIRECT VARIABLE SIZE            =   40H+02H     64D+  2D
    INDIRECT VARIABLE SIZE          =   40H+00H     64D+  0D
    BIT SIZE                        =   01H+00H      1D+  0D
    BIT-ADDRESSABLE SIZE            =   00H+00H      0D+  0D
    AUXILIARY VARIABLE SIZE         = 0093H       147D
    MAXIMUM STACK SIZE              = 0019H        25D
    REGISTER-BANK(S) USED:            0 1
    456 LINES READ
    0 PROGRAM ERROR(S)
END OF PL/M-51 COMPILATION
```

296166-68

RUPI™ Datasheet

PRELIMINARY

8044AH/8344AH/8744H
HIGH PERFORMANCE 8-BIT MICROCONTROLLER WITH ON-CHIP SERIAL COMMUNICATION CONTROLLER

- **8044AH—Includes Factory Mask Programmable ROM**
- **8344AH—For Use with External Program Memory**
- **8744H—Includes User Programmable/Eraseable EPROM**

8051 MICROCONTROLLER CORE

- **Optimized for Real Time Control 12 MHz Clock, Priority Interrupts, 32 Programmable I/O Lines, Two 16-bit Timer/Counters**
- **Boolean Processor**
- **4K × 8 ROM, 192 × 8 RAM**
- **64K Accessible External Program Memory**
- **64K Accessible External Data Memory**
- **4 μs Multiply and Divide**

SERIAL INTERFACE UNIT (SIU)

- **Serial Communication Processor that Operates Concurrently to CPU**
- **2.4 Mbps Maximum Data Rate**
- **375 Kbps using On-Chip Phase Locked Loop**
- **Communication Software in Silicon:**
  — Complete Data Link Functions
  — Automatic Station Response
- **Operates as an SDLC Primary or Secondary Station**

The RUPI-44 family integrates a high performance 8-bit Microcontroller, the Intel 8051 Core, with an Intelligent/high performance HDLC/SDLC serial communication controller, called the Serial Interface Unit (SIU). See Figure 1. This dual architecture allows complex control and high speed data communication functions to be realized cost effectively.

Specifically, the 8044's Microcontroller features: 4K byte On-Chip program memory space; 32 I/O lines; two 16-bit timer/event counters; a 5-source; 2-level interrupt structure; a full duplex serial channel; a Boolean processor; and on-chip oscillator and clock circuitry. Standard TTL and most byte-oriented MCS-80 and MCS-85 peripherals can be used for I/O amd memory expansion.

The Serial Interface Unit (SIU) manages the interface to a high speed serial link. The SIU offloads the On-Chip 8051 Microcontroller of communication tasks, thereby freeing the CPU to concentrate on real time control tasks.

The RUPI-44 family consists of the 8044, 8744, and 8344. All three devices are identical except in respect of on-chip program memory. The 8044 contains 4K bytes of mask-programmable ROM. User programmable EPROM replaces ROM in the 8744. The 8344 addresses all program memory externally.

The RUPI-44 devices are fabricated with Intel's reliable +5 volt, silicon-gate HMOSII technology and packaged in a 40-pin DIP.

The 8744H is available in a hermetically sealed, ceramic, 40-lead dual in-line package which includes a window that allows for EPROM erasure when exposed to ultraviolet light (See Erasure Characteristics). During normal operation, ambient light may adversely affect the functionality of the chip. Therefore applications which expose the 8744H to ambient light may require an opaque label over the window.

8044's Dual Controller Architecture

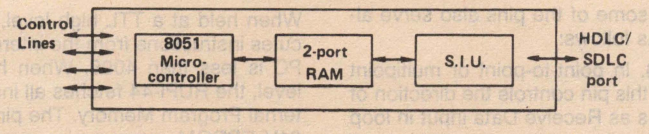


Figure 1. Dual Controller Architecture

November 1986
Order Number: 231663-003

Table 1. RUPI™-44 Family Pin Description

VSS

Circuit ground potential.

VCC

+5V power supply during operation and program verification.

PORT 0

Port 0 is an 8-bit open drain bidirectional I/O port. It is also the multiplexed low-order address and data bus when using external memory. It is used for data output during program verification. Port 0 can sink/source eight LS TTL loads.

PORT 1

Port 1 is an 8-bit quasi-bidirectional I/O port. It is used for the low-order address byte during program verification. Port 1 can sink/source four LS TTL loads.

In non-loop mode two of the I/O lines serve alternate functions:

— $\overline{RTS}$ (P1.6). Request-to-Send output. A low indicates that the RUPI-44 is ready to transmit.

— $\overline{CTS}$ (P1.7) Clear-to-Send input. A low indicates that a receiving station is ready to receive.

PORT 2

Port 2 is an 8-bit quasi-bidirection I/O port. It also emits the high-order address byte when accessing external memory. It is used for the high-order address and the control signals during program verification. Port 2 can sink/source four LS TTL loads.

PORT 3

Port 3 is an 8-bit quasi-bidirectional I/O port. It also contains the interrupt, timer, serial port and RD and WR pins that are used by various options. The output latch corresponding to a secondary function must be programmed to a one (1) for that function to operate. Port 3 can sink/source four LS LTT loads.

In addition to I/O, some of the pins also serve alternate functions as follows:

— I/$\overline{O}$ RxD (P3.0). In point-to-point or multipoint configurations, this pin controls the direction of pin P3.1. Serves as Receive Data input in loop and diagnostic modes.

— DATA TxD (P3.1) In point-to-point or multipoint configurations, this pin functions as data input/output. In loop mode, it serves as transmit pin. A '0' written to this pin enables diagnostic mode.

— $\overline{INT0}$ (P3.2). Interrupt 0 input or gate control input for counter 0.

— $\overline{INT1}$ (P3.3). Interrupt 1 input or gate control input for counter 1.

— TO (P3.4). Input to counter 0.

— SCLK T1 (P3.5). In addition to I/O, this pin provides input to counter 1 or serves as SCLK (serial clock) input.

— $\overline{WR}$ (P3.6). The write control signal latches the data byte from Port 0 into the External Data Memory.

— $\overline{RD}$ (P3.7). The read control signal enables External Data Memory to Port 0.

RST

A high on this pin for two machine cycles while the oscillator is running resets the device. A small external pulldown resistor ($\approx$8.2KΩ) from RST to V_{SS} permits power-on reset when a capacitor ($\approx$10μf) is also connected from this pin to V_{CC}.

ALE/$\overline{PROG}$

Provides Address Latch Enable output used for latching the address into external memory during normal operation. It is activated every six oscillator periods except during an external data memory access. It also receives the program pulse input for programming the EPROM version.

$\overline{PSEN}$

The Program Store Enable output is a control signal that enables the external Program Memory to the bus during external fetch operations. It is activated every six oscillator periods, except during external data memory accesses. Remains high during internal program execution.

$\overline{EA}$/VPP

When held at a TTL high level, the RUPI-44 executes instructions from the internal ROM when the PC is less than 4096. When held at a TTL low level, the RUPI-44 fetches all instructions from external Program Memory. The pin also receives the 21V EPROM programming supply voltage on the 8744.

Table 1. RUPI™-44 Family Pin Description (Continued)

XTAL 1	XTAL 2
Input to the oscillator's high gain amplifier. Required when a crystal is used. Connect to VSS when external source is used on XTAL 2.	Output from the oscillator's amplifier. Input to the internal timing circuitry. A crystal or external source can be used.

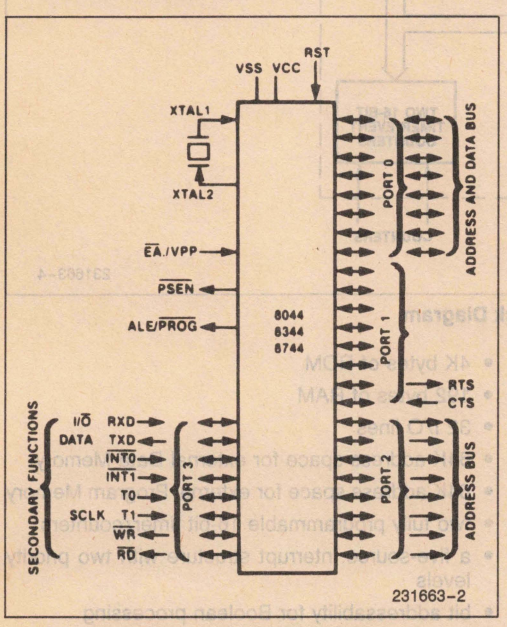


Figure 2. Logic Symbol

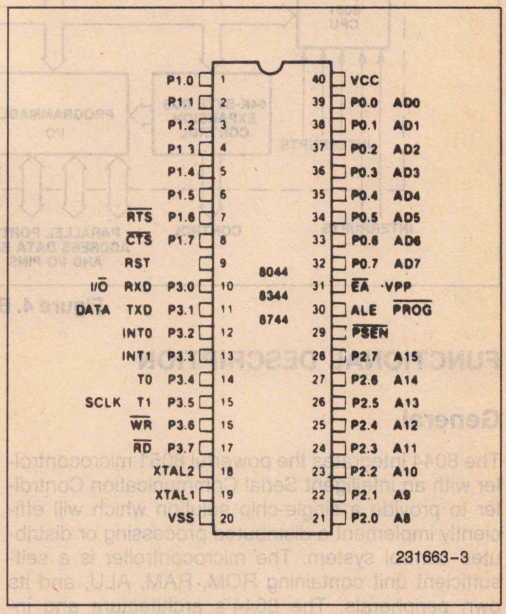


Figure 3A. DIP Pin Configuration

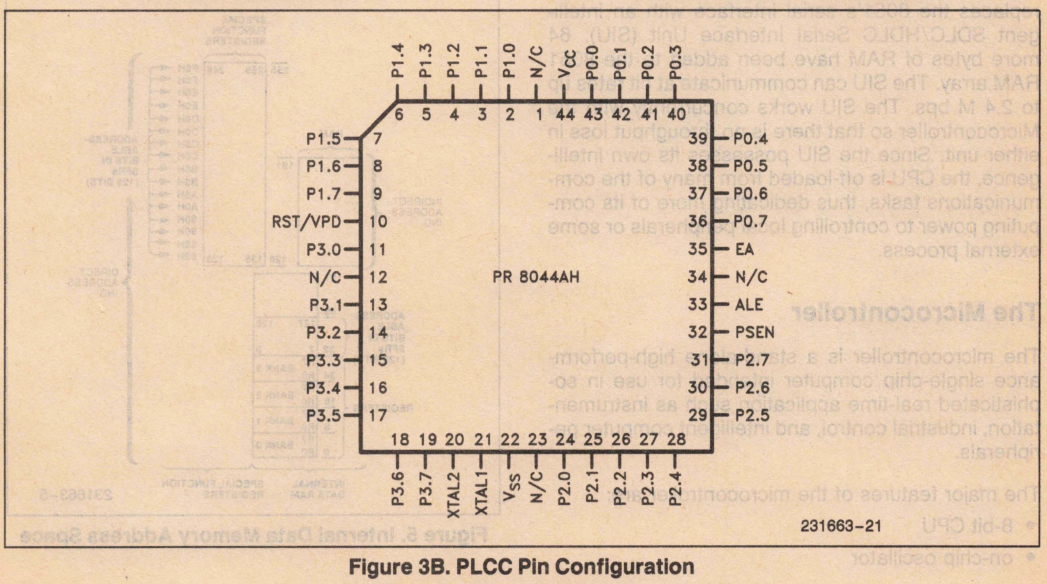


Figure 3B. PLCC Pin Configuration

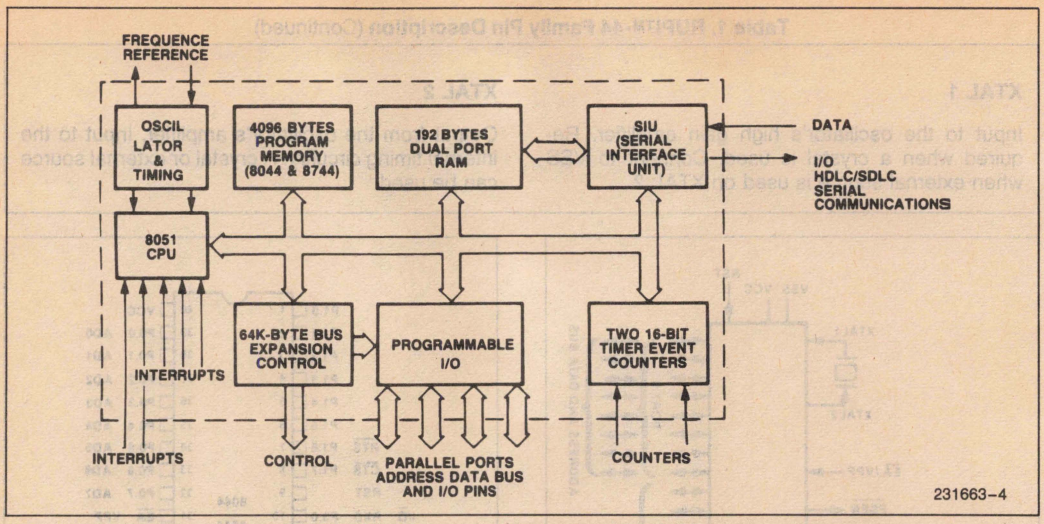


Figure 4. Block Diagram

FUNCTIONAL DESCRIPTION

General

The 8044 integrates the powerful 8051 microcontroller with an intelligent Serial Communication Controller to provide a single-chip solution which will efficiently implement a distributed processing or distributed control system. The microcontroller is a self-sufficient unit containing ROM, RAM, ALU, and its own peripherals. The 8044's architecture and instruction set are identical to the 8051's. The 8044 replaces the 8051's serial interface with an intelligent SDLC/HDLC Serial Interface Unit (SIU). 64 more bytes of RAM have been added to the 8051 RAM array. The SIU can communicate at bit rates up to 2.4 M bps. The SIU works concurrently with the Microcontroller so that there is no throughput loss in either unit. Since the SIU possesses its own intelligence, the CPU is off-loaded from many of the communications tasks, thus dedicating more of its computing power to controlling local peripherals or some external process.

The Microcontroller

The microcontroller is a stand-alone high-performance single-chip computer intended for use in sophisticated real-time application such as instrumentation, industrial control, and intelligent computer peripherals.

The major features of the microcontroller are:

- 8-bit CPU
- on-chip oscillator
- 4K bytes of ROM
- 192 bytes of RAM
- 32 I/O lines
- 64K address space for external Data Memory
- 64K address space for external Program Memory
- two fully programmable 16-bit timer/counters
- a five-source interrupt structure with two priority levels
- bit addressability for Boolean processing

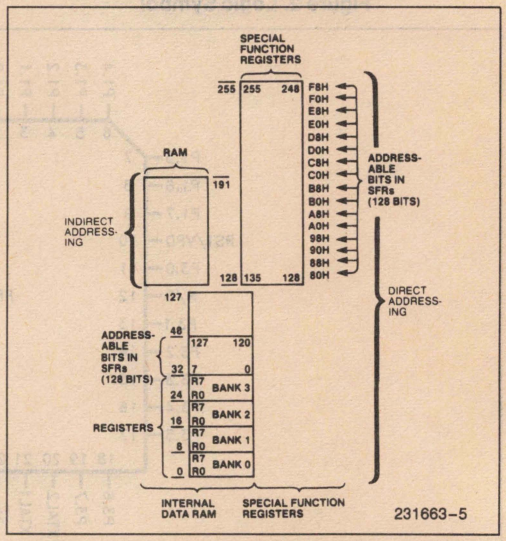


Figure 5. Internal Data Memory Address Space

- 1 μs instruction cycle time for 60% of the instructions 2 μs instruction cycle time for 40% of the instructions
- 4 μs cycle time for 8 by 8 bit unsigned Multiply/Divide

INTERNAL DATA MEMORY

Functionally the Internal Data Memory is the most flexible of the address spaces. The Internal Data Memory space is subdivided into a 256-byte Internal Data RAM address space and a 128-bit Special Function Register address space as shown in Figure 5.

The Internal Data RAM address space is 0 to 255. Four 8-Register Banks occupy locations 0 through 31. The stack can be located anywhere in the Internal Data RAM address space. In addition, 128 bit locations of the on-chip RAM are accessible through Direct Addressing. These bits reside in Internal Data RAM at byte locations 32 through 47. Currently locations 0 through 191 of the Internal Data RAM address space are filled with on-chip RAM.

Parallel I/O

The 8044 has 32 general-purpose I/O lines which are arranged into four groups of eight lines. Each group is called a port. Hence there are four ports; Port 0, Port 1, Port 2, and Port 3. Up to five lines from 1 and Port 2 are dedicated to supporting the serial channel when the SIU is invoked. Due to the nature of the serial port, two of Port 3's I/O lines (P3.0 and P3.1) do not have latched outputs. This is true whether or not the serial channel is used.

Port 0 and Port 2 also have an alternate dedicated function. When placed in the external access mode, Port 0 and Port 2 become the means by which the 8044 communicates with external program memory. Port 0 and Port 2 are also the means by which the 8044 communicates with external data memory. Peripherals can be memory mapped into the address space and controlled by the 8044.

Table 2. MCS®-51 Instruction Set Description

Mnemonic		Description	Byte	Cyc
ARITHMETIC OPERATIONS				
ADD	A,Rn	Add register to Accumulator	1	1
ADD	A,direct	Add direct byte to Accumulator	2	1
ADD	A,@Ri	Add indirect RAM to Accumulator	1	1
ADD	A,#data	Add immediate data to Accumulator	2	1
ADDC	A,Rn	Add register to Accumulator with Carry	1	1
ADDC	A,direct	Add direct byte to A with Carry flag	2	1
ADDC	A,@Ri	Add indirect RAM to A with Carry flag	1	1
ADDC	A,#data	Add immediate data to A with Carry flag	2	1
SUBB	A,Rn	Subtract register from A with Borrow	1	1
SUBB	A,direct	Subtract direct byte from A with Borrow	2	1
SUBB	A,@Ri	Subtract indirect RAM from A with Borrow	1	1
SUBB	A,#data	Subtract immed data from A with Borrow	2	1
INC	A	Increment Accumulator	1	1
INC	Rn	Increment register	1	1
INC	direct	Increment direct byte	2	1
INC	@Ri	Increment indirect RAM	1	1
INC	DPTR	Increment Data Pointer	1	2
DEC	A	Decrement Accumulator	1	1
DEC	Rn	Decrement register	1	1
DEC	direct	Decrement direct byte	2	1
DEC	@Ri	Decrement indirect RAM	1	1
MUL	AB	Multiply A & B	1	4
DIV	AB	Divide A by B	1	4
DA	A	Decimal Adjust Accumulator	1	1

Table 2. MCS®-51 Instruction Set Description (Continued)

Mnemonic		Description	Byte	Cyc
LOGICAL OPERATIONS				
ANL	A,Rn	AND register to Accumulator	1	1
ANL	A,direct	AND direct byte to Accumulator	2	1
ANL	A,@RI	AND indirect RAM to Accumulator	1	1
ANL	A,#data	AND immediate data to Accumulator	2	1
ANL	direct,A	AND Accumulator to direct byte	2	1
ANL	direct,#data	AND immediate data to direct byte	3	2
ORL	A,Rn	OR register to Accumulator	1	1
ORL	A,direct	OR direct byte to Accumulator	2	1
ORL	A,@Ri	OR indirect RAM to Accumulator	1	1
ORL	A,#data	OR immediate data to Accumulator	2	1
ORL	direct,A	OR Accumulator to direct byte	2	1
ORL	direct,#data	OR immediate data to direct byte	3	2
XRL	A,Rn	Exclusive-OR register to Accumulator	1	1
XRL	A,direct	Exclusive-OR direct byte to Accumulator	2	1
XRL	A,@RI	Exclusive-OR indirect RAM to A	1	1
XRL	A,#data	Exclusive-OR immediate data to A	2	1
XRL	direct,A	Exclusive-OR Accumulator to direct byte	2	1
XRL	direct,#data	Exclusive-OR immediate data to direct	3	2
CLR	A	Clear Accumulator	1	1
CPL	A	Complement Accumulator	1	1

Mnemonic		Description	Byte	Cyc
LOGICAL OPERATIONS (Continued)				
RL	A	Rotate Accumulator Left	1	1
RLC	A	Rotate A Left through the Carry flag	1	1
RR	A	Rotate Accumulator Right	1	1
RRC	A	Rotate A Right through Carry flag	1	1
SWAP	A	Swap nibbles within the Accumulator	1	1
DATA TRANSFER				
MOV	A,Rn	Move register to Accumulator	1	1
MOV	A,direct	Move direct byte to Accumulator	2	1
MOV	A,@RI	Move indirect RAM to Accumulator	1	1
MOV	A,#data	Move immediate data to Accumulator	2	1
MOV	Rn,A	Move Accumulator to register	1	1
MOV	Rn,direct	Move direct byte to register	2	2
MOV	Rn,#data	Move immediate data to register	2	1
MOV	direct,A	Move Accumulator to direct byte	2	1
MOV	direct,Rn	Move register to direct byte	2	2
MOV	direct,direct	Move direct byte to direct	3	2
MOV	direct,@Ri	Move indirect RAM to direct byte	2	2
MOV	direct,#data	Move immediate data to direct byte	3	2
MOV	@Ri,A	Move Accumulator to indirect RAM	1	1
MOV	@Ri,direct	Move direct byte to indirect RAM	2	2

Table 2. MCS®-51 Instruction Set Description (Continued)

Mnemonic		Description	Byte	Cyc
DATA TRANSFER (Continued)				
MOV	@Ri,#data	Move immediate data to indirect RAM	2	1
MOV	DPTR,#data16	Load Data Pointer with a 16-bit constant	3	2
MOVC	A,@A+DPTR	Move Code byte relative to DPTR to A	1	2
MOVC	A,@A+PC	Move Code byte relative to PC to A	1	2
MOVX	A,@Ri	Move External RAM (8-bit addr) to A	1	2
MOVX	A,@DPTR	Move External RAM (16-bit addr) to A	1	2
MOVX	@Ri,A	Move A to External RAM (8-bit addr)	1	2
MOVX	@DPTR,A	Move A to External RAM (16-bit) addr	1	2
PUSH	direct	Push direct byte onto stack	2	2
POP	direct	Pop direct byte from stack	2	2
XCH	A,Rn	Exchange register with Accumulator	1	1
XCH	A,direct	Exchange direct byte with Accumulator	2	1
XCH	A,@Ri	Exchange indirect RAM with A	1	1
XCHD	A,@Ri	Exchange low-order Digit ind RAM w A	1	1
BOOLEAN VARIABLE MANIPULATION				
CLR	C	Clear Carry flag	1	1
CLR	bit	Clear direct bit	2	1
SETB	C	Set Carry Flag	1	1
SETB	bit	Set direct Bit	2	1
CPL	C	Complement Carry Flag	1	1
CPL	bit	Complement direct bit	2	1
ANL	C,bit	AND direct bit to Carry flag	2	2
ANL	C,/bit	AND complement of direct bit to Carry	2	2
ORL	C/bit	OR direct bit to Carry flag	2	2
ORL	C,/bit	OR complement of direct bit to Carry	2	2
MOV	C,/bit	Move direct bit to Carry flag	2	1
MOV	bit,C	Move Carry flag to direct bit	2	2
PROGRAM AND MACHINE CONTROL				
ACALL	addr11	Absolute Subroutine Call	2	2
LCALL	addr16	Long Subroutine Call	3	2
RET		Return from subroutine	1	2
RETI		Return from interrupt	1	2
AJMP	addr11	Absolute Jump	2	2
LJMP	addr16	Long Jump	3	2
SJMP	rel	Short Jump (relative addr)	2	2
JMP	@A+DPTR	Jump indirect relative to the DPTR	1	2
JZ	rel	Jump if Accumulator is Zero	2	2
JNZ	rel	Jump if Accumulator is Not Zero	2	2
JC	rel	Jump if Carry flag is set	2	2
JNC	rel	Jump if No Carry flag	2	2
JB	bit,rel	Jump if direct Bit set	3	2
JNB	bit,rel	Jump if direct Bit Not set	3	2
JBC	bit,rel	Jump if direct Bit is set & Clear bit	3	2
CJNE	A,direct,rel	Compare direct to A & Jump if Not Equal	3	2
CJNE	A,#data,rel	Comp, immed, to A & Jump if Not Equal	3	2

Table 2. MCS®-51 Instruction Set Description (Continued)

Mnemonic	Description	Byte	Cyc
PROGRAM AND MACHINE CONTROL (Continued)			
CJNE Rn,#data,rel	Comp, immed, to reg & Jump if Not Equal	3	2
CJNE @Ri,#data, rel	Comp, immed, to ind. & Jump if Not Equal	3	2
DJNZ Rn,rel	Decrement register & Jump if Not Zero	2	2
DJNZ direct,rel	Decrement direct & Jump if Not Zero	3	2
NOP	No operation	1	1

Notes on data addressing modes:

Rn — Working register R0-R7

direct — 128 internal RAM locations, any I/O port, control or status register

@Ri — Indirect internal RAM location addressed by register R0 or R1

Notes on data addressing modes:
(Continued)

#data — 8-bit constant included in instruction

#data16 — 16-bit constant included as bytes 2 & 3 of instruction

bit — 128 software flags, any I/O pin, controll or status bit

Notes on program addressing modes:

addr16 — Destination address for LCALL & LJMP may be anywhere within the 64-K program memory address space

Addr11 — Destination address for ACALL & AJMP will be within the same 2-K page of program memory as the first byte of the following instruction

rel — SJMP and all conditional jumps include an 8-bit offset byte, Range is +127 −128 bytes relative to first byte of the following instruction

All mnemonic copyrighted© Intel Corporation 1979

Timer/Counters

The 8044 contains two 16-bit counters which can be used for measuring time intervals, measuring pulse widths, counting events, generating precise periodic interrupt requests, and clocking the serial communications. Internally the Timers are clocked at 1/12 of the crystal frequency, which is the instruction cycle time. Externally the counters can run up to 500 KHz.

Interrupt System

External events and the real-time driven on-chip peripherals require service by the CPU asynchronous to the execution of any particular section of code. To tie the asynchronous activities of these functions to normal program execution, a sophisticated multiple-source, two priority level, nested interrupt system is provided. Interrupt response latency ranges from 3 μsec to 7 μsec when using a 12 MHz clock.

All five interrupt sources can be mapped into one of the two priority levels. Each interrupt source can be enabled or disabled individually or the entire interrupt system can be enabled or disabled. The five interrupt sources are: Serial Interface Unit, Timer 1, Timer 2, and two external interrupts. The external interrupts can be either level or edge triggered.

Serial Interface Unit (SIU)

The Serial Interface Unit is used for HDLC/SDLC communications. It handles Zero Bit Insertion/Deletion, Flags automatic access recognization, and a 16-bit cyclic redundancy check. In addition it implements in hardware a subset of the SDLC protocol certain applications it is advantageous to have the CPU control the reception or transmission of every single frame. For this reason the SIU has two modes of operation: "AUTO" and "FLEXIBLE" (or "NON-AUTO"). It is in the AUTO mode that the SIU responds to SDLC frames without CPU intervention; whereas, in the FLEXIBLE mode the reception or transmission of every single frame will be under CPU control.

There are three control registers and eight parameter registers that are used to operate the serial interface. These registers are shown in Figure 5 and Figure 6. The control register set the modes of operation and provide status information. The eight parameter registers buffer the station address, receive and transmit control bytes, and point to the on-chip transmit and receive buffers.

Data to be received or transmitted by the SIU must be buffered anywhere within the 192 bytes of on-chip RAM. Transmit and receive buffers are not allowed to "wrap around" in RAM; a "buffer end" is generated after address 191 is reached.

REGISTER NAMES	SYMBOLIC ADDRESS	BIT ADDRESS	BYTE ADDRESS	
B REGISTER	B	247 through 240	240	(F0H)
ACCUMULATOR	ACC	231 through 224	224	(E0H)
*THREE BYTE FIFO	FIFO		223	(DFH)
	FIFO		222	(DEH)
	FIFO		221	(DDH)
TRANSMIT BUFFER START	TBS		220	(DCH)
TRANSMIT BUFFER LENGTH	TBL		219	(DBH)
TRANSMIT CONTROL BYTE	TCB		218	(DAH)
* SIU STATE COUNTER	SIUST		217	(D9H)
SEND COUNT RECEIVE COUNT	NSNR	223 through 216	216	(D8H)
PROGRAM STATUS WORD	PSW	215 through 208	208	(D0H)
*DMA COUNT	DMA CNT		207	(CFH)
STATION ADDRESS	STAD		206	(CEH)
RECEIVE FIELD LENGTH	RFL		205	(CDH)
RECEIVE BUFFER START	RBS		204	(CCH)
RECEIVE BUFFER LENGTH	RBL		203	(CBH)
RECEIVE CONTROL BYTE	RCB		202	(CAH)
SERIAL MODE	SMD		201	(C9H)
STATUS REGISTER	STS	207 through 200	200	(C8H)
INTERRUPT PRIORITY CONTROL	IP	191 through 184	184	(B8H)
PORT 3	P3	183 through 176	176	(B0H)
INTERRUPT ENABLE CONTROL	IE	175 through 168	168	(A8H)
PORT 2	P2	167 through 160	160	(A0H)
PORT 1	P1	151 through 144	144	(90H)
TIMER HIGH 1	TH1		141	(8DH)
TIMER HIGH 0	TH0		140	(8CH)
TIMER LOW 1	TL1		139	(8BH)
TIMER LOW 0	TL0		138	(8AH)
TIMER MODE	TMOD		137	(89H)
TIMER CONTROL	TCON	143 through 136	136	(88H)
DATA POINTER HIGH	DPH		131	(83H)
DATA POINTER LOW	DPL		130	(82H)
STACK POINTER	SP		129	(81H)
PORT 0	P0	135 through 128	128	(80H)

SFR's CONTAINING DIRECT ADDRESSABLE BITS

231663-6

NOTE:
*ICE Support Hardware registers. Under normal operating conditions there is no need for the CPU to access these registers.

Figure 5. Mapping of Special Function Registers

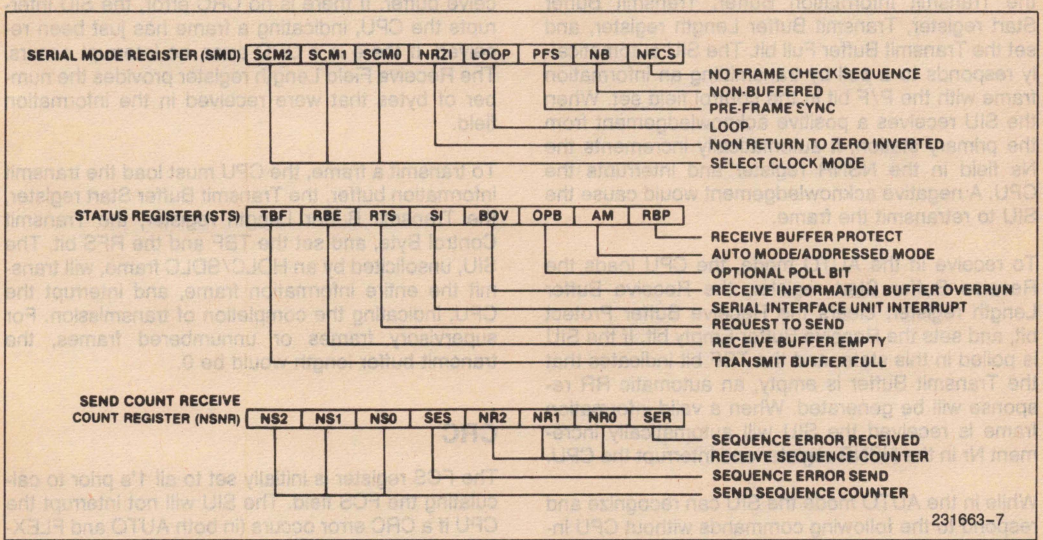


Figure 6. Serial Interface Unit Control Registers

With the addition of only a few bytes of code, the 8044's frame size is not limited to the size of its internal RAM (192 bytes), but rather by the size of external buffer with no degradation of the RUPI's features (e.g. NRZI, zero bit insertion/deletion, address recognition, cyclic redundancy check). There is a special function register called SIUST whose contents dictates the operation of the SIU. At low data rates, one section of the SIU (the Byte Processor) performs no function during known intervals. For a given data rate, these intervals (stand-by mode) are fixed. The above characteristics make it possible to program the CPU to move data to/from external RAM and to force the SIU to perform some desired hardware tasks while transmission or reception is taking place. With these modifications, external RAM can be utilized as a transmit and received buffer instead of the internal RAM.

AUTO Mode

In the AUTO mode the SIU implements in hardware a subset of the SDLC protocol such that it responds to many SDLC frames without CPU intervention. All AUTO mode responses to the primary station will comform to IBM's SDLC definition. The advantages of the AUTO mode are that less software is required to implement a secondary station, and the hardware generated response to polls is much faster than doing it in software. However, the Auto mode can not be used at a primary station.

To transmit in the AUTO mode the CPU must load the Transmit Information Buffer, Transmit Buffer Start register, Transmit Buffer Length register, and set the Transmit Buffer Full bit. The SIU automatically responds to a poll by transmitting an information frame with the P/F bit in the control field set. When the SIU receives a positive acknowledgement from the primary station, it automatically increments the Ns field in the NSNR register and interrupts the CPU. A negative acknowledgement would cause the SIU to retransmit the frame.

To receive in the AUTO mode, the CPU loads the Receive Buffer Start register, the Receive Buffer Length register, clears the Receive Buffer Protect bit, and sets the Receive Buffer Empty bit. If the SIU is polled in this state, and the TBF bit indicates that the Transmit Buffer is empty, an automatic RR response will be generated. When a valid information frame is received the SIU will automatically increment Nr in the NSNR register and interrupt the CPU.

While in the AUTO mode the SIU can recognize and respond to the following commands without CPU intervention: I (Information), RR (Receive Ready), RNR (Receive Not Ready), REJ (Reject), and UP (Unnumbered Poll). The SIU can generate the following responses without CPU intervention: I (Information), RR (Receive Ready), and RNR (Receive Not Ready).

When the Receive Buffer Empty bit (RBE) indicates that the Receive Buffer is empty, the receiver is enabled, and when the RBE bit indicates that the Receive Buffer is full, the receiver is disabled. Assuming that the Receiver Buffer is empty, the SIU will respond to a poll with an I frame if the Transmit Buffer is full. If the Transmit Buffer is empty, the SIU will respond to a poll with a RR command if the Receive Buffer Protect bit (RBP) is cleared, or an RNR command if RBP is set.

FLEXIBLE (or NON-AUTO) Mode

In the FLEXIBLE mode all communications are under control of the CPU. It is the CPU's task to encode and decode control fields, manage acknowledgements, and adhere to the requirements of the HDLC/SDLC protocols. The 8044 can be used as a primary or a secondary station in this mode.

To receive a frame in the FLEXIBLE mode, the CPU must load the Receive Buffer Start register, the Receive Buffer Length register, clear the Receive Buffer Protect bit, and set the Receive Buffer Empty bit. If a valid opening flag is received and the address field matches the byte in the Station Address register or the address field contains a broadcast address, the 8044 loads the control field in the receive control byte register, and loads the I field in the receive buffer. If there is no CRC error, the SIU interrupts the CPU, indicating a frame has just been received. If there is a CRC error, no interrupt occurs. The Receive Field Length register provides the number of bytes that were received in the information field.

To transmit a frame, the CPU must load the transmit information buffer, the Transmit Buffer Start register, the Transmit Buffer Length register, the Transmit Control Byte, and set the TBF and the RFS bit. The SIU, unsolicited by an HDLC/SDLC frame, will transmit the entire information frame, and interrupt the CPU, indicating the completion of transmission. For supervisory frames or unnumbered frames, the transmit buffer length would be 0.

CRC

The FCS register is initially set to all 1's prior to calculating the FCS field. The SIU will not interrupt the CPU if a CRC error occurs (in both AUTO and FLEXIBLE modes). The CRC error is cleared upon receiving of an opening flag.

Frame Format Options

In addition to the standard SDLC frame format, the 8044 will support the frames displayed in Figure 7. The standard SDLC frame is shown at the top of this figure. For the remaining frames the information field will incorporate the control or address bytes and the frame check sequences; therefore these fields will be stored in the Transmit and Receive buffers. For example, in the non-buffered mode the third byte is treated as the beginning of the information field. In the non-addressed mode, the information field begins after the opening flag. The mode bits to set the frame format options are found in the Serial Mode register and the Status register.

FRAME OPTION	NFCS	NB	AM[1]	FRAME FORMAT
Standard SDLC NON-AUTO Mode	0	0	0	F A C I FCS F
Standard SDLC AUTO Mode	0	0	1	F A C I FCS F
Non-Buffered Mode NON-AUTO Mode	0	1	1	F A I FCS F
Non-Addressed Mode NON-AUTO Mode	0	1	0	F I FCS F
No FCS Field NON-AUTO Mode	1	0	0	F A C I F
No FCS Field AUTO Mode	1	0	1	F A C I F
No FCS Field Non-Buffered Mode NON-AUTO Mode	1	1	1	F A I F
No FCS Field Non-Addressed Mode NON-AUTO Mode	1	1	0	F I F

Mode Bits:
AM — "AUTO" Mode/Addressed Mode
NB — Non-Buffered Mode
NFCS — No FCS Field Mode

Key to Abbreviations:
F = Flag (01111110) I = Information Field
A = Address Field FCS= Frame Check Sequence
C = Control Field

Note 1:
The AM bit function is controlled by the NB bit. When NB = 0, AM becomes AUTO mode select, when NB = 1, AM becomes Address mode select.

Figure 7. Frame Format Options

Extended Addressing

To realize an extended control field or an extended address field using the HDLC protocol, the FLEXIBLE mode must be used. For an extended control field, the SIU is programmed to be in the non-buffered mode. The extended control field will be the first and second bytes in the Receive and Transmit Buffers. For extended addressing the SIU is placed in the non-addressed mode. In this mode the CPU must implement the address recognition for received frames. The addressing field will be the initial bytes in the Transmit and Receive buffers followed by the control field.

The SIU can transmit and receive only frames which are multiples of 8 bits. For frames received with other than 8-bit multiples, a CRC error will cause the SIU to reject the frame.

SDLC Loop Networks

The SIU can be used in an SDLC loop as a secondary or primary station. When the SIU is placed in the Loop mode it receives the data on pin 10 and transmits the data one bit time delayed on pin 11. It can also recognize the Go ahead signal and change it into a flag when it is ready to transmit. As a secondary station the SIU can be used in the AUTO or FLEXIBLE modes. As a primary station the FLEXIBLE mode is used; however, additional hardware is required for generating the Go Ahead bit pattern. In the Loop mode the maximum data rate is 1 Mbps clocked or 375 Kpbs self-clocked.

SDLC Multidrop Networks

The SIU can be used in a SDLC non-loop configuration as a secondary or primary station. When the SIU is placed in the non-loop mode, data is received and transmitted on pin 11, and pin 10 drives a tri-state buffer. In non-loop mode, modem interface pins, RTS and CTS, become available.

Data Clocking Options

The 8044's serial port can operate in an externally clocked or self clocked system. A clocked system provides to the 8044 a clock synchronization to the data. A self-clocked system uses the 8044's on-chip Digital Phase Locked Loop (DPLL) to recover the clock from the data, and clock this data into the Serial Receive Shift Register.

In this mode, a clock synchronized with the data is externally fed into the 8044. This clock may be generated from an External Phase Locked Loop, or possibly supplied along with the data. The 8044 can transmit and receive data in this mode at rates up to 2.4 Mbps.

This self clocked mode allows data transfer without a common system data clock. An on-chip Digital Phase Locked Loop is employed to recover the data clock which is encoded in the data stream. The DPLL will converge to the nominal bit center within eight bit transitions, worst case. The DPLL requires a reference clock of either 16 times (16x) or 32 times (32x) the data rate. This reference clock may be externally applied or internally generated. When internally generated either the 8044's internal logic clock (crystal frequency divided by two) or the timer 1 overflow is used as the reference clock. Using the internal timer 1 clock the data rates can vary from 244 to 62.5 Kbps. Using the internal logic clock at a 16x sampling rate, receive data can either be 187.5 Kbps, or 375 Kbps. When the reference clock for the DPLL is externally applied the data rates can vary from 0 to 375 Kbps at a 16x sampling rate.

To aid in a Phase Locked Loop capture, the SIU has a NRZI (Non Return to Zero Inverted) data encoding and decoding option. Additionally the SIU has a preframe sync option that transmits two bytes of alternating 1's and 0's to ensure that the receive station DPLL will be synchronized with the data by the time it receives the opening flag.

Control and Status Registers

There are three SIU Control and Status Registers:

Serial Mode Register (SMD)

Status/Command Register (STS)

Send/Receive Count Register (NSNR)

The SMD, STS, and NSNR, registers are all cleared by system reset. This assures that the SIU will power up in an idle state (neither receiving nor transmitting).

These registers and their bit assignments are described below.

SMD: Serial Mode Register (byte-addressable)

Bit 7:	6	5	4	3	2	1	0
SCM2	SCM1	SCM0	NRZI	LOOP	PFS	NB	NFCS

The Serial Mode Register (Address C9H) selects the operational modes of the SIU. The 8044 CPU can both read and write SMD. The SIU can read SMD but cannot write to it. To prevent conflict between CPU and SIU access to SMD, the CPU should write SMD only when the Request To Send (RTS) and

Receive Buffer Empty (RBE) bits (in the STS register) are both false (0). Normally, SMD is accessed only during initialization.

The individual bits of the Serial Mode Register are as follows:

Bit #	Name	Description
SMD.0	NFCS	No FCS field in the SDLC frame.
SMD.1	NB	Non-Buffered mode. No control field in the SDLC frame.
SMD.2	PFS	Pre-Frame Sync mode. In this mode, the 8044 transmits two bytes before the first flag of a frame, for DPLL synchronization. If NRZI is enabled, 00H is sent; otherwise, 55H is sent. In either case, 16 preframe transitions are guaranteed.
SMD.3	LOOP	Loop configuration.
SMD.4	NRZI	NRZI coding option. If bit = 1, NRZI coding is used. If bit = 0, then it is straight binary (NRZ).
SMD.5	SCM0	Select Clock Mode—Bit 0
SMD.6	SCM1	Select Clock Mode—Bit 1
SMD.7	SCM2	Select Clock Mode—Bit 2

The SCM bits decode as follows:

SCM				Data Rate
2	1	0	Clock Mode	(Bits/sec)*
0	0	0	Externally clocked	0-2.4M**
0	0	1	Reserved	
0	1	0	Self clocked, timer overflow	244-62.5K
0	1	1	Reserved	
1	0	0	Self clocked, external 16x	0-375K
1	0	1	Self clocked, external 32x	0-187.5K
1	1	0	Self clocked, internal fixed	375K
1	1	1	Self clocked, internal fixed	187.5K

NOTES:
*Based on a 12 Mhz crystal frequency
**0-1 M bps in loop configuration

STS: Status/Command Register (bit-addressable)

Bit:	7	6	5	4	3	2	1	0
	TBF	RBE	RTS	SI	BOV	OPB	AM	RBP

The Status/Command Register (Address C8H) provides operational control of the SIU by the 8044 CPU, and enables the SIU to post status information for the CPU's access. The SIU can read STS, and can alter certain bits, as indicated below. The CPU can both read and write STS asynchronously. However, 2-cycle instructions that access STS during both cycles ('JBC/B, REL' and 'MOV/B, C.') should not be used, since the SIU may write to STS between the two CPU accesses.

The individual bits of the Status/Command Register are as follows:

Bit #	Name	Description
STS.0	RBP	Receive Buffer Protect. Inhibits writing of data into the receive buffer. In AUTO mode, RBP forces an RNR response instead of an RR.
STS.1	AM	AUTO Mode/Addressed Mode. Selects AUTO mode where AUTO mode is allowed. If NB is true, (= 1), the AM bit selects the addressed mode. AM may be cleared by the SIU.
STS.2	OPB	Optional Poll Bit. Determines whether the SIU will generate an AUTO response to an optional poll (UP with P = 0). OPM may be set or cleared by the SIU.
STS.3	BOV	Receive Buffer Overrun. BOV may be set or cleared by the SIU.
STS.4	SI	SIU Interrupt. This is one of the five interrupt sources to the CPU. The vector location = 23H. SI may be set by the SIU. It should be cleared by the CPU before returning from an interrupt routine.
STS.5	RTS	Request To Send. Indicates that the 8044 is ready to transmit or is transmitting. RTS may be read or written by the CPU. RTS may be read by the SIU, and in AUTO mode may be written by the SIU.
STS.6	RBE	Receive Buffer Empty. RBE can be thought of as Receive Enable. RBE is set to one by the CPU when it is ready to receive a frame, or has just read the buffer, and to zero by the SIU when a frame has been received.
STS.7	TBF	Transmit Buffer Full. Written by the CPU to indicate that it has filled the transmit buffer. TBF may be cleared by the SIU.

NSNR: Send/Receive Count Register (bit-addressable)

Bit:	7	6	5	4	3	2	1	0
	NS2	NS1	NS0	SES	NR2	NR1	NR0	SER

The Send/Receive Count Register (Address D8H) contains the transmit and receive sequence numbers, plus tally error indications. The SIU can both read and write NSNR. The 8044 CPU can both read and write NSNR asynchronously. However, 2-cycle instructions that access NSNR during both cycles ('JBC /B, REL,' and 'MOV /B,C') should not be used, since the SIU may write to NSMR between the two 8044 CPU accesses.

The individual bits of the Send/Receive Count Register are as follows:

Bit #	Name	Description
NSNR.0	SER	Receive Sequence Error: NS (P) ≠ NR (S)
NSNR.1	NR0	Receive Sequence Counter—Bit 0
NSNR.2	NR1	Receive Sequence Counter—Bit 1
NSNR.3	NR2	Receive Sequence Counter—Bit 2
NSNR.4	SES	Send Sequence Error: NR (P) ≠ NS (S) and NR (P) ≠ NS (S) + 1
NSNR.5	NS0	Send Sequence Counter—Bit 0
NSNR.6	NS1	Send Sequence Counter—Bit 1
NSNR.7	NS2	Send Sequence Counter—Bit 2

Parameter Registers

There are eight parameter registers that are used in connection with SIU operation. All eight registers may be read or written by the 8044 CPU. RFL and RCB are normally loaded by the SIU.

The eight parameter registers are as follows:

STAD: Station Address Register (byte-addressable)

The Station Address register (Address CEH) contains the station address. To prevent acess conflict, the CPU should access STAD only when the SIU is idle (RTS = 0 and RBE = 0). Normally, STAD is accessed only during initialization.

TBS: Transmit Buffer Start Address Register (byte-addressable)

The Transmit Buffer Start address register (Address DCH) points to the location in on-chip RAM for the beginning of the I-field of the frame to be transmitted. The CPU should access TBS only when the SIU is not transmitting a frame (when TBF = 0).

TBL: Transmit Buffer Length Register (byte = addressable)

The Transmit Buffer Length register (Address DBH) contains the length (in bytes) of the I-field to be transmitted. A blank I-field (TBL = 0) is valid. The CPU should access TBL only when the SIU is not transmitting a frame (when TBF = 0).

NOTE:
The transmit and receive buffers are not allowed to "wrap around" in the on-chip RAM. A "buffer end" is automatically generated if address 191 (BFH) is reached.

TCB: Transmit Control Byte Register (byte-addressable)

The Transmit Control Byte register (Address DAH) contains the byte which is to be placed in the control field of the transmitted frame, during NON-AUTO mode transmission. The CPU should access TCB only when the SIU is not transmitting a frame (when TBF = 0). The N_S and N_R counters are not used in the NON-AUTO mode.

RBS: Receive Buffer Start Address Register (byte-addressable)

The Receive Buffer Start address register (Address CCH) points to the location in on-chip RAM where the beginning of the I-field of the frame being received is to be stored. The CPU should write RBS only when the SIU is not receiving a frame (when RBE = 0).

RBL: Receive Buffer Length Register (byte-addressable)

The Receive Buffer Length register (Address CBH) contains the length (in bytes) of the area in on-chip RAM allocated for the received I-field. RBL=0 is valid. The CPU should write RBL only when RBE=0.

RFL: Receive Field Length Register (byte-addressable)

The Receive Field Length register (Address CDH) contains the length (in bytes) of the received I-field that has just been loaded into on-chip RAM. RFL is loaded by the SIU. RFL = 0 is valid. RFL should be accessed by the CPU only when RBE = 0.

RCB: Receive Control Byte Register (byte-addressable)

The Received Control Byte register (Address CAH) contains the control field of the frame that has just been received. RCB is loaded by the SIU. The CPU can only read RCB, and should only access RCB when RBE = 0.

ICE Support

The 8044 In-Circuit Emulator (ICE-44) allows the user to exercise the 8044 application system and monitor the execution of instructions in real time.

The emulator operates with Intel's Intellec™ development system. The development system interfaces with the user's 8044 system through an in-cable buffer box. The cable terminates in a 8044 pin-compatible plug, which fits into the 8044 socket in the user's system. With the emulator plug in place, the user can excercise his system in real time while collecting up to 255 instruction cycles of real-time data. In addition, he can single-step the program.

Static RAM is available (in the in-cable buffer box) to emulate the 8044 internal and external program memory and external data memory. The designer can display and alter the contents of the replacement memory in the buffer box, the internal data memory, and the internal 8044 registers, including the SFR's.

SIUST: SIU State Counter (byte-addressable)

The SIU State Counter (Address D9H) reflects the state of the internal logic which is under SIU control. Therefore, care must be taken not to write into this register. This register provides a useful means for debugging 8044 receiver problem.

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias . . . . . 0°C to 70°C
Storage Temperature . . . . . −65°C to −150°C
Voltage on $\overline{EA}$, VPP Pin to VSS . . . −0.5V to −21.5V
Voltage on Any Other Pin to VSS . . . . −0.5V to −7V
Power Dissipation 2W

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS T_A = 0°C to 70°C, VCC = 5V ± 10%, VSS = 0V

Symbol	Parameter		Min	Max	Unit	Test Conditions
VIL	Input Low Voltage (Except $\overline{EA}$ Pin of 8744H)		−0.5	0.8	V	
VIL1	Input Low Voltage to $\overline{EA}$ Pin of 8744H		0	0.8	V	
VIH	Input High Voltage (Except XTAL2, RST)		2.0	VCC + 0.5	V	
VIH1	Input High Voltage to XTAL2, RST		2.5	VCC + 0.5	V	XTAL1 = VSS
VOL	Output Low Voltage (Ports 1, 2, 3)*			0.45	V	IOL = 1.6mA
VOL1	Output Low Voltage (Port 0,ALE,$\overline{PSEN}$)*					
		8744H		0.60 0.45	V V	IOL = 3.2 mA IOL = 2.4 mA
		8044AH/8344AH		0.45	V	IOL = 3.2 mA
VOH	Output High Voltage (Ports 1, 2, 3)		2.4		V	IOH = −80 μA
VOH1	Output High Voltage (Port 0 in External Bus Mode, ALE, $\overline{PSEN}$)		2.4		V	IOH = −400 μA
IIL	Logical 0 Input Current (Ports 1, 2, 3)			−500	μA	Vin = 0.45V
IIL1	Logical 0 Input Current to $\overline{EA}$ Pin of 8744H only			−15	mA	
IIL2	Logical 0 Input Current (XTAL2)			−3.6	mA	Vin = 0.45V
ILI	Input Leakage Current (Port 0) 8744H 8044AH/8344AH			 ±100 ±10	 μA μA	 0.45 < Vin < VCC 0.45 < Vin < VCC
IIH	Logical 1 Input Current to $\overline{EA}$ Pin of 8744H			500	μA	
IIH1	Input Current to RST to Activate Reset			500	μA	Vin < (VCC − 1.5V)
ICC	Power Supply Current: 8744H 8044AH/8344AH			 285 170	 mA mA	All Outputs Disconnected: $\overline{EA}$ = VCC
CIO	Pin Capacitance			10	pF	Test Freq. = 1MHz

***NOTES:**
Capacitive loading on Ports 0 and 2 may cause spurious noise pulses to be superimposed on the VOLs of ALE and Ports 1 and 3. The noise is due to external bus capacitance discharging into the Port 0 and Port 2 pin when these pins make 1-to-0 transitions during bus operations. In the worst cases (capacitive loading > 100 pF), the noise pulse on the ALE line may exceed 0.8V. In such cases it may be desirable to qualify ALE with a Schmitt Trigger, or use an address latch with a Schmitt Trigger STROBE input.

A.C. CHARACTERISTICS

T_A = 0°C to +70°C, VCC = 5V ± 10%, VSS = 0V, Load Capacitance for Port 0, ALE, and PSEN = 100 pF, Load Capacitance for All Other Outputs = 80 pF

EXTERNAL PROGRAM MEMORY CHARACTERISTICS

Symbol	Parameter	12 MHz Osc		Variable Clock 1/TCLCL = 1.2 MHz to 12 MHz		Unit
		Min	Max	Min	Max	
TLHLL	ALE Pulse Width	127		2TCLCL-40		ns
TAVLL	Address Valid to ALE Low	43		TCLCL-40		ns
TLLAX[1]	Address Hold After ALE Low	48		TCLCL-35		ns
TLLIV	ALE Low to Valid Instr in					ns
	8744H		183		4TCLCL-150	
	8044AH/8344AH		233		4TCLCL-100	
TLLPL	ALE Low to $\overline{\text{PSEN}}$ Low	58		TCLCL-25		ns
TPLPH	$\overline{\text{PSEN}}$ Pulse Width					
	8744H	190		3TCLCL-60		ns
	8044AH/8344AH	215		3TCLCL-35		ns
TPLIV	$\overline{\text{PSEN}}$ Low to Valid Instr in					
	8744H		100		3TCLCL-150	ns
	8044AH/8344AH		125		3TCLCL-125	ns
TPXIX	Input Instr Hold After $\overline{\text{PSEN}}$	0		0		ns
TPXIZ[2]	Input Instr Float After $\overline{\text{PSEN}}$		63		TCLCL-20	ns
TPXAV[2]	$\overline{\text{PSEN}}$ to Address Valid	75		TCLCL-8		ns
TAVIV	Address to Valid Instr in					
	8744H		267		5TCLCL-150	ns
	8044AH/8344AH		302		5TCLCL-115	ns
TAZPL	Address Float to $\overline{\text{PSEN}}$	−25		−25		ns

NOTES:
1. TLLAX for access to program memory is different from TLLAX for data memory.
2. Interfacing RUPI-44 devices with float times up to 75ns is permissible. This limited bus contention will not cause any damage to Port 0 drivers.

EXTERNAL DATA MEMORY CHARACTERISTICS

Symbol	Parameter	12 MHz Osc		Variable Clock 1/TCLCL = 1.2 MHz to 12 MHz		Unit
		Min	Max	Min	Max	
TRLRH	$\overline{RD}$ Pulse Width	400		6TCLCL-100		ns
TWLWH	$\overline{WR}$ Pulse Width	400		6TCLCL-100		ns
TLLAX	Address Hold after ALE	48		TCLCL-35		ns
TRLDV	$\overline{RD}$ Low to Valid Data in		252		5TCLCL-165	ns
TRHDX	Data Hold After $\overline{RD}$	0		0		ns
TRHDZ	Data Float After $\overline{RD}$		97		2TCLCL-70	ns
TLLDV	ALE Low to Valid Data In		517		8TCLCL-150	ns
TAVDV	Address to Valid Data In		585		9TCLCL-165	ns
TLLWL	ALE Low to $\overline{RD}$ or $\overline{WR}$ Low	200	300	3TCLCL-50	3TLCLCL+50	ns
TAVWL	Address to $\overline{RD}$ or $\overline{WR}$ Low	203		4TCLCL-130		ns
TQVWX	Data Valid to $\overline{WR}$ Transition					
	8744H	13		TCLCL-70		ns
	8044AH/8344AH	23		TCLCL-60		ns
TQVWH	Data Setup Before $\overline{WR}$ High	433		7TCLCL-150		ns
TWHQX	Data Held After $\overline{WR}$	33		TCLCL-50		ns
TRLAZ	$\overline{RD}$ Low to Address Float		25		25	ns
TWHLH	$\overline{RD}$ or $\overline{WR}$ High to ALE High					
	8744H	33	133	TCLCL-50	TCLCL+50	ns
	8044AH/8344AH	43	123	TCLCL-40	TCLCL+50	ns

NOTE:
1. TLLAX for access to program memory is different from TLLAX for access data memory.

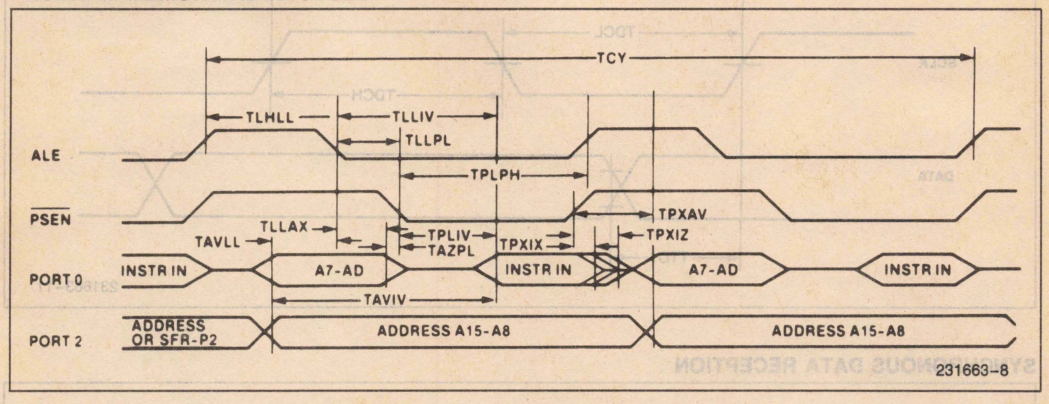
Serial Interface

Symbol	Parameter	Min	Max	Units
TDCY	Data Clock	420		ns
TDCL	Data Clock Low	180		ns
TDCH	Data Clock High	100		ns
tTD	Transmit Data Delay		140	ns
tDSS	Data Setup Time	40		ns
tDHS	Data Hold Time	40		ns

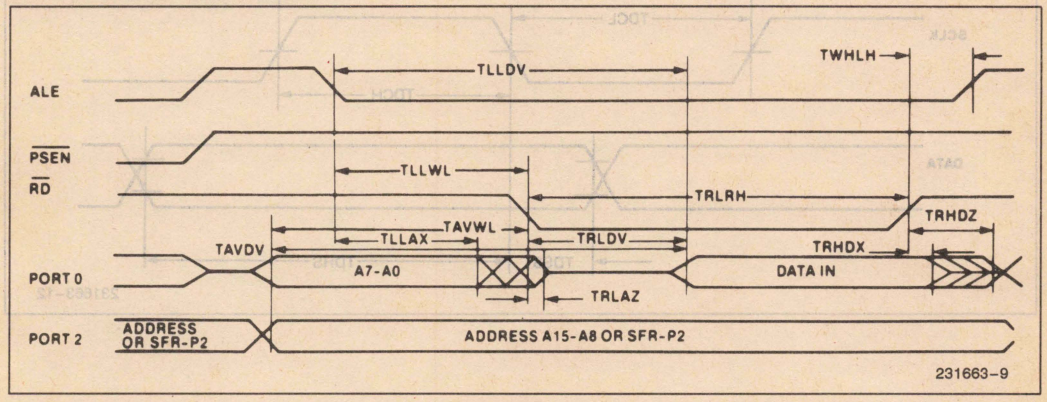
WAVEFORMS

Memory Access

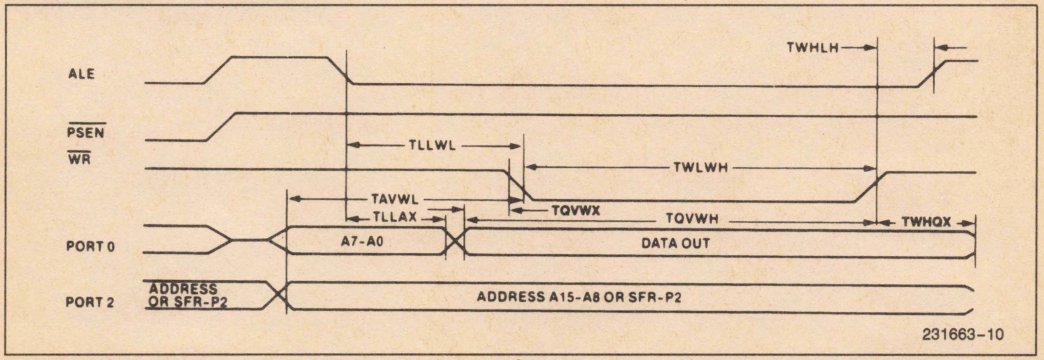
PROGRAM MEMORY READ CYCLE



DATA MEMORY READ CYCLE

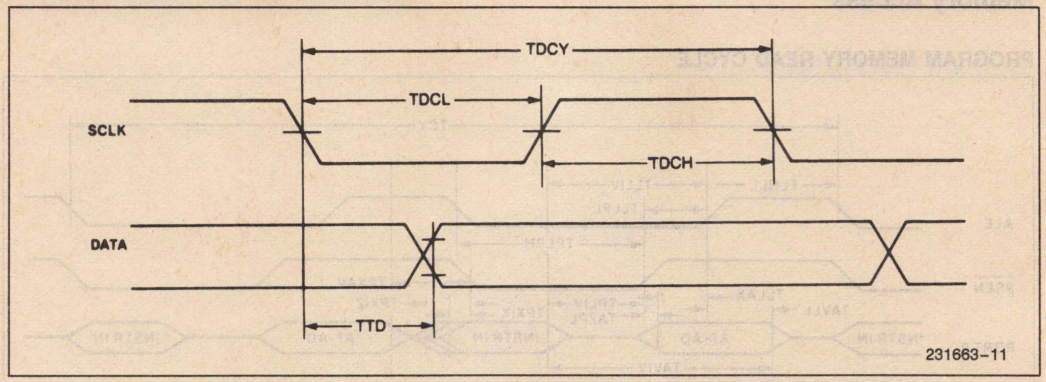


DATA MEMORY WRITE CYCLE

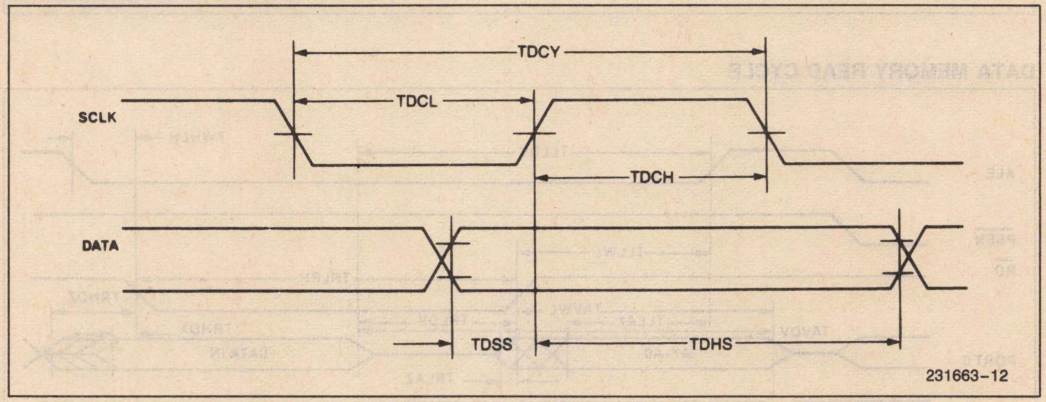


SERIAL I/O WAVEFORMS

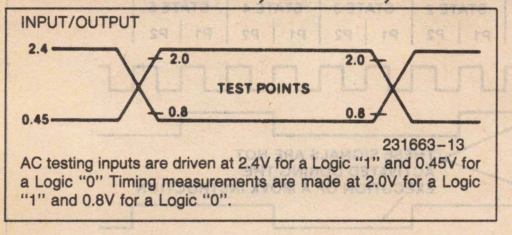
SYNCHRONOUS DATA TRANSMISSION



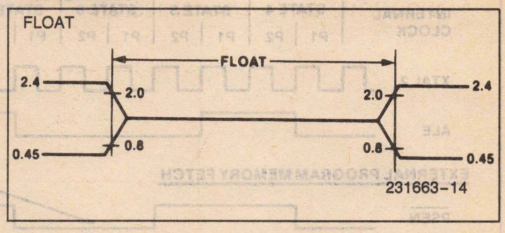
SYNCHRONOUS DATA RECEPTION



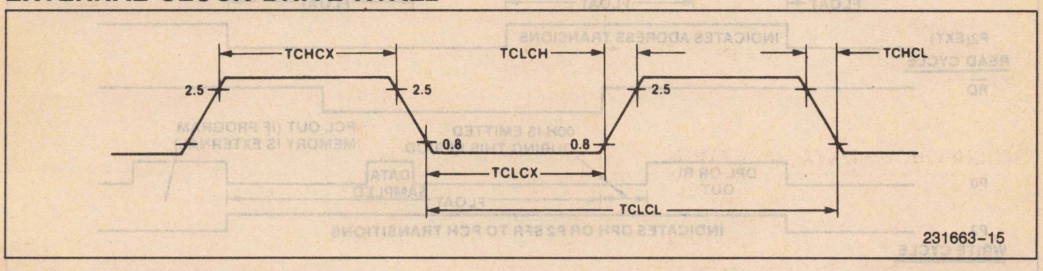
AC TESTING INPUT, OUTPUT, FLOAT WAVEFORMS



AC testing inputs are driven at 2.4V for a Logic "1" and 0.45V for a Logic "0" Timing measurements are made at 2.0V for a Logic "1" and 0.8V for a Logic "0".

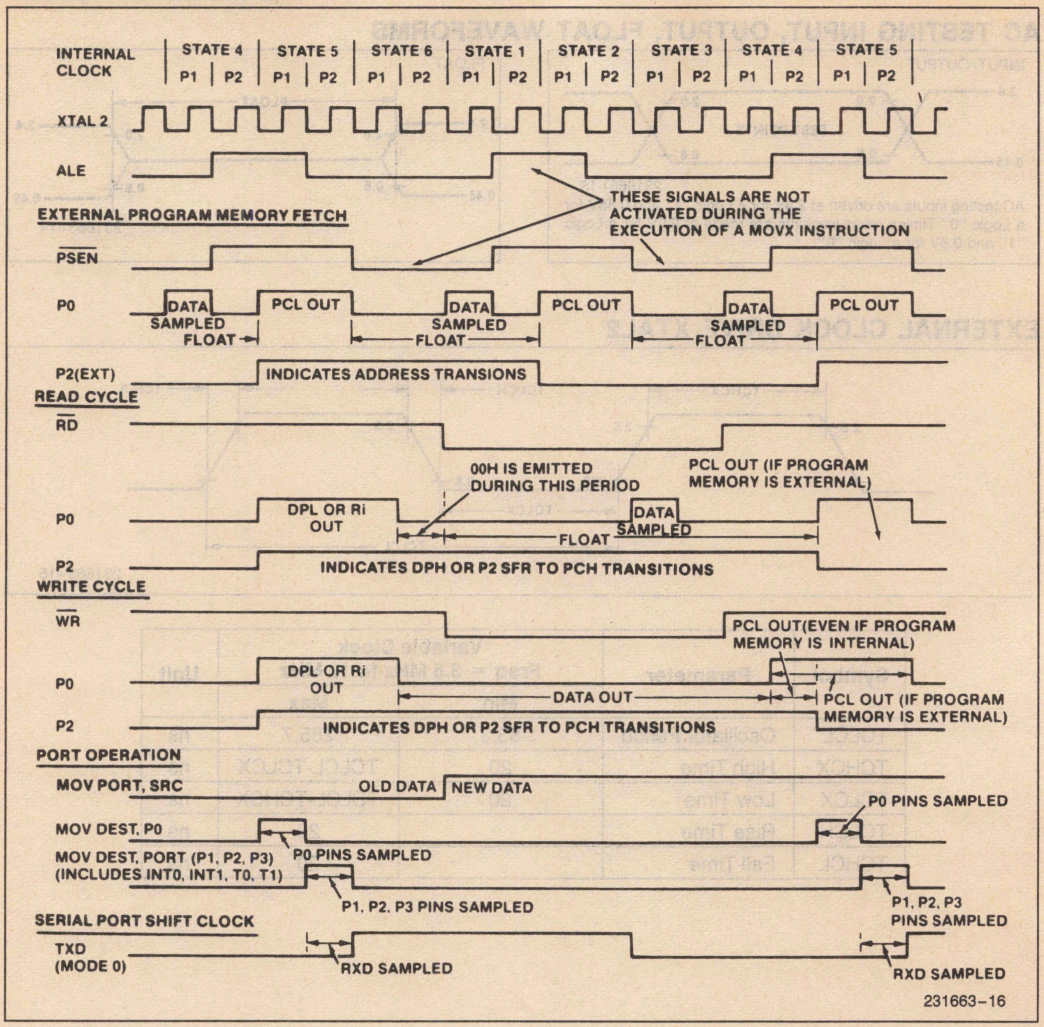


EXTERNAL CLOCK DRIVE XTAL2



Symbol	Parameter	Variable Clock Freq = 3.5 MHz to 12 MHz		Unit
		Min	Max	
TCLCL	Oscillator Period	83.3	285.7	ns
TCHCX	High Time	20	TCLCL-TCLCX	ns
TCLCX	Low Time	20	TCLCL-TCHCX	ns
TCLCH	Rise Time		20	ns
TCHCL	Fall Time		20	ns

CLOCK WAVEFORMS



This diagram indicates when signals are clocked internally. The time it takes the signals to propagate to the pins, however, ranges from 25 to 125 ns. This propagation delay is dependent on variables such as temperature and pin loading. Propagation also varies from output to output and component to component. Typically though, (T_A = 25°C, fully loaded) RD and WR propagation delays are approximately 50 ns. The other signals are typically 85 ns. Propagation delays are incorporated in the AC specifications.

8744H EPROM CHARACTERISTICS

Erasure Characteristics

Erasure of the 8744H Program Memory begins to occur when the chip is exposed to light with wavelengths shorter than approximately 4,000 Ångstroms. Since sunlight and fluorescent lighting have wavelengths in this range, constant exposure to these light sources over an extended period of time (about 1 week in sunlight, or 3 years in room-level fluorescent lighting) could cause unintentional erasure. If an application subjects the 8744H to this type of exposure, it is suggested that an opaque label be placed over the window.

The recommended erasure procedure is exposure to ultraviolet light (at 2537 Ångstroms) to an integrated dose of at least 15 W-sec/cm^2 rating for 20 to 30 minutes, at a distance of about 1 inch, should be sufficient.

Erasure leaves the array in an all 1s state.

Programming the EPROM

To be programmed, the 8744H must be running with a 4 to 6 MHz oscillator. (The reason the oscillator needs to be running is that the internal bus is being used to transfer address and program data to appropriate registers.) The address of an EPROM location to be programmed is applied to Port 1 and pins P2.0-P2.3 of Port 2, while the data byte is applied to Port 0. Pins P2.4-P2.6 and $\overline{PSEN}$ should be held low, and P2.7 and RST high. (These are all TTL levels except RST, which requires 2.5V for high.) $\overline{EA}$/VPP is held normally high, and is pulsed to +21V. While EA/VPP is at 21V, the ALE/$\overline{PROG}$ pin, which is normally being held high, is pulsed low for 50 msec. Then EA/VPP is returned to high. This is illustrated in Figure 3. Detailed timing specifications are provided in the EPROM Programming and Verification Characteristics section of this data sheet.

Program Memory Security

The program memory security feature is developed around a "security bit" in the 8744H EPROM array. Once this "hidden bit" is programmed, electrical access to the contents of the entire program memory array becomes impossible. Activation of this feature is accomplished by programming the 8744H as described in "Programming the EPROM" with the exception that P2.6 is held at a TTL high rather than a TTL low. In addition, Port 1 and P2.0-P2.3 may be in any state. Figure 4 illustrates the security bit programming configuration. Deactivating the security feature, which again allows programmability of the EPROM, is accomplished by exposing the EPROM to ultraviolet light. This exposure, as described in "Erasure Characteristics," erases the entire EPROM array. Therefore, attempted retrieval of "protected code" results in its destruction.

Program Verification

Program Memory may be read only when the "security feature" has not been activated. Refer to Figure 5 for Program Verification setup. To read the Program Memory, the following procedure can be used. The unit must be running with a 4 to 6 MHz oscillator. The address of a Program Memory location to be read is applied to Port 1 and pins P2.0-P2.3 of Port 2. Pins P2.4-P2.6 and $\overline{PSEN}$ are held at TTL low, while the ALE/$\overline{PROG}$, RST, and $\overline{EA}$/VPP pins are held at TTL high. (These are all TTL levels except RST, which requires 2.5V for high.) Port 0 will be the data output lines. P2.7 can be used as a read strobe. While P2.7 is held high, the Port 0 pins float. When P2.7 is strobed low, the contents of the addressed location will appear at Port 0. External pullups (e.g., 10K) are required on Port 0 during program verification.

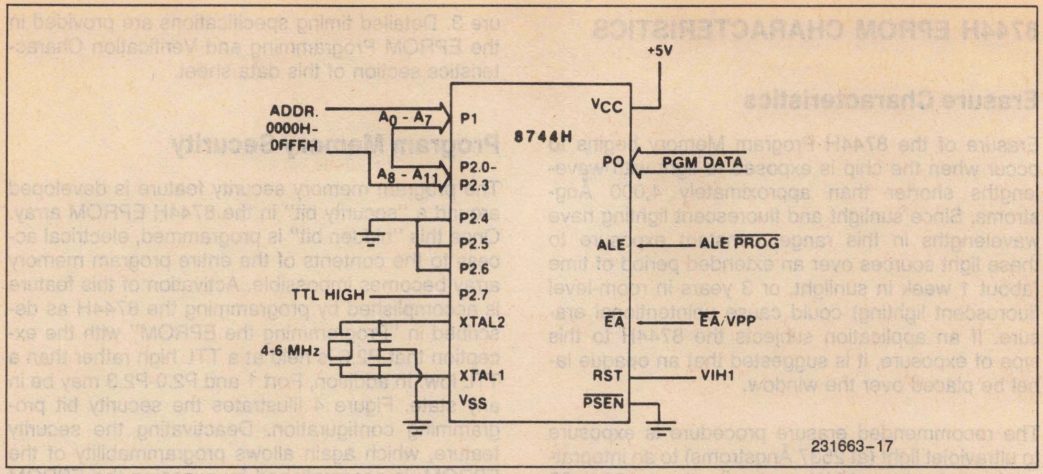


Figure 3. Programming Configuration

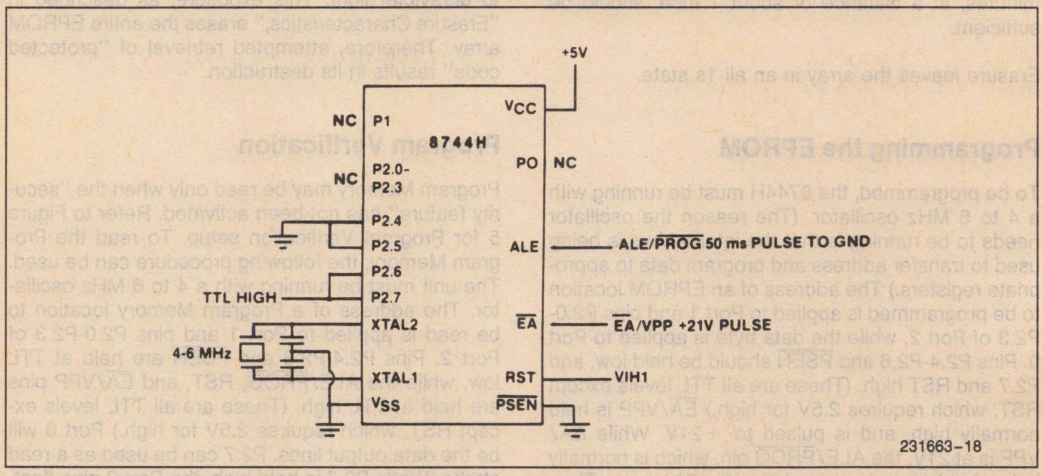


Figure 4. Security Bit Programming Configuration

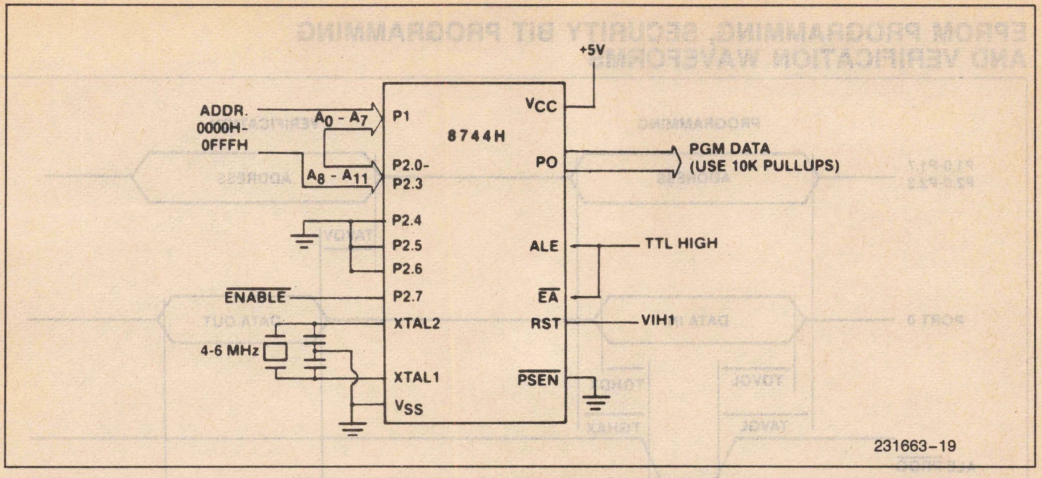


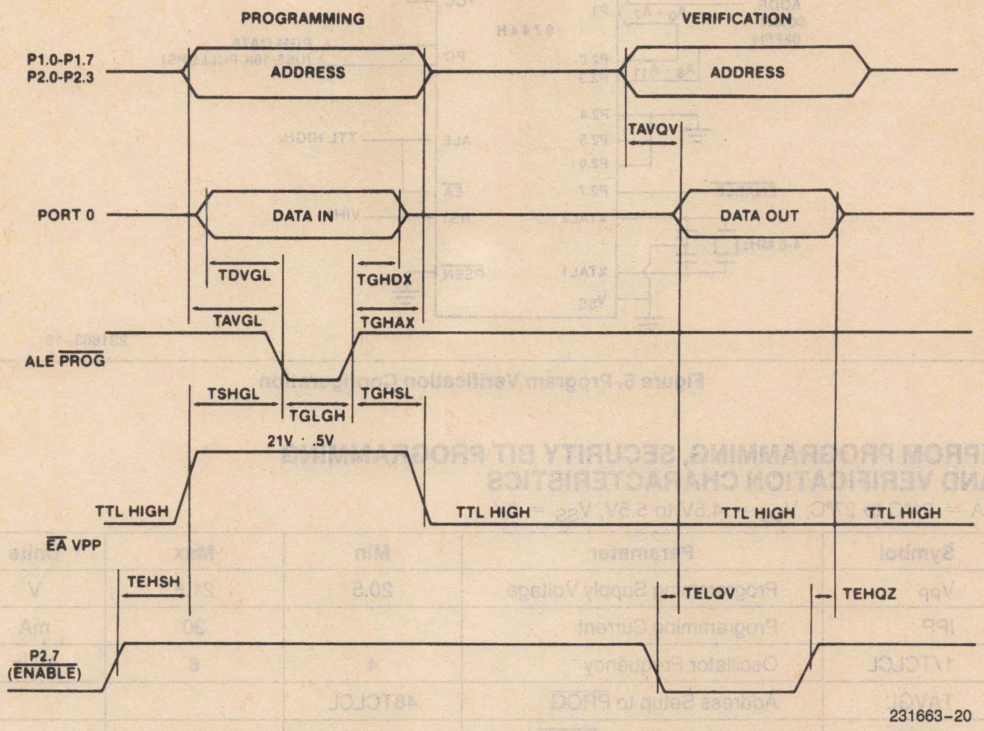
Figure 5. Program Verification Configuration

EPROM PROGRAMMING, SECURITY BIT PROGRAMMING AND VERIFICATION CHARACTERISTICS

TA = 21°C to 27°C, V_{CC} = 4.5V to 5.5V, V_{SS} = 0V

Symbol	Parameter	Min	Max	Units
V_{PP}	Programming Supply Voltage	20.5	21.5	V
IPP	Programming Current		30	mA
1/TCLCL	Oscillator Frequency	4	6	MHz
TAVGL	Address Setup to $\overline{PROG}$	48TCLCL		
TGHAX	Address Hold after $\overline{PROG}$	48TCLCL		
TDVGL	Data Setup to $\overline{PROG}$	48TCLCL		
TGHDX	Data Hold after $\overline{PROG}$	48TCLCL		
TEHSH	$\overline{ENABLE}$ High to Vpp	48TCLCL		
TSHGL	V_{PP} Setup to $\overline{PROG}$	10		μsec
TGHSL	V_{PP} Hold after $\overline{PROG}$	10		μsec
TGLGH	$\overline{PROG}$ Width	45	55	msec
TAVQV	Address to Data Valid		48TCLCL	
TELQV	$\overline{ENABLE}$ to Data Valid		48TCLCL	
TEHQZ	Data Float after $\overline{ENABLE}$	0	48TCLCL	

EPROM PROGRAMMING, SECURITY BIT PROGRAMMING AND VERIFICATION WAVEFORMS



For programming conditions see Figure 3.
For security bit programming conditions see Figure 4.
For verification conditions see Figure 5.

RUPI™ Article Reprint 24

RUPI™ Article Reprint 24

intel

ARTICLE REPRINT

AR-307

NOVEMBER 1983

MICROCONTROLLER WITH INTEGRATED HIGH PERFORMANCE COMMUNICATIONS INTERFACE

CHARLES YAGER
APPLICATIONS ENGINEER

January 1985

© INTEL CORPORATION

ORDER NUMBER 230876-001

230876-1

SUMMARY

The 8044 offers a lower cost and higher performance solution to networking microcontrollers than conventional solutions. The system cost is lowered by integrating an entire microcomputer with an intelligent HDLC/SDLC communication processor onto a single chip. The higher performance is realized by integrating two processors running concurrently on one chip; the powerful 8051 microcontroller and the Serial Interface Unit. The 8051 microcontroller is substantially off-loaded from the communication tasks when using the AUTO mode. In the AUTO mode the SIU handles many of the data link functions in hardware. The advantages of the AUTO mode are: less software is required to implement a secondary station data link, the 8051 CPU is offloaded, and the turn-around time is reduced, thus increasing the network throughput. Currently the 8044 is the only microcontroller with a sophisticated communications processor on-chip. In the future there will be more microcontrollers available following this trend.

INTRODUCTION

Today microcontrollers are being designed into virtually every type of equipment. For the household, they are turning up in refrigerators, thermostats, burglar alarms, sprinklers, and even water softeners. At work they are found in laboratory instruments, copiers, elevators, hospital equipment, and telephones. In addition, a lot of microcomputer equipment contains more than one microcontroller. Applications using multiple microcontrollers as well, as the office and home, are now faced with the same requirements that laboratory instruments were faced with 12 years ago — they need to connect them together and have them communicate. This need was satisfied in the laboratory with the IEEE-488 General Purpose Instrumentation Bus (GPIB). However, GPIB does not meet the current design objectives for networking microcontrollers.

Today there are many communications schemes and protocols available; some of the popular ones are GPIB, Async, HDLC/SDLC, and Ethernet. Common design objectives of today's networks are: low cost, reliable, efficient throughput, and expandable. In examining available solutions, GPIB does not meet these design objectives; first, the cable is too expensive (parallel communications), second, it can only be used over a limited distance (20 meters), and third, it can only handle a limited number of stations. For general networking, serial communications is preferable because of lower cable costs and higher reliability (fewer connections). While Ethernet provides very high performance, it is more of a system backbone rather than a microcontroller interconnect. Async, on the other hand, is inexpensive but it is not an efficient protocol for data block or file transfers. Even with some new modifications such as a 9 bit protocol for addressing, important functions such as acknowledgements, error checking/recovery, and data transparency are not standardized nor supported by available data comm chips.

SDLC, Synchronous Data Link Control, meets the requirements for communications link design. The physical medium can be used on two or four wire twisted pair with inexpensive transceivers and connectors. It can also be interfaced through modems, which allows it to be used on broadband networks, leased or switched telephone lines. VLSI controllers have been available from a number of vendors for years; higher performance and more user friendly SDLC controllers continue to appear. SDLC has also been designed to be very reliable. A 16 bit CRC checks the integrity of the received data, while frame numbering and acknowledgements are also built in. Using SDLC, up to 254 stations can be uniquely addressed, while HDLC addressing is unlimited. If an RS-422 only requires a single +5 volt power supply.

What will the end user pay for the added value provided by communications? The cost of the communications hardware is not the only additional cost. There will be performance degradation in the main application because the microcontroller now has additional tasks to perform. There are two extremes to the cost of adding communication capability. One could spend very little by adding an I/O port and have the CPU handle everything from the baud rate to the protocol. Of course the main application would be idle while the CPU was communicating. The other extreme would be to add another microcontroller to the system dedicated to communications. This communications processor could interface to the main CPU through a high speed parallel link or dual port RAM. This approach would maintain system performance, but it would be costly.

Adding HDLC/SDLC Networking Capability

Figure 1 shows a microcomputer system with a conventional HDLC SDLC communications solution. The additional hardware needed to realize the conventional design is: an HDLC SDLC communication chip, additional ROM for the communication software, part of an interrupt controller, a baud rate generator, a phase locked loop, NRZI encoded decoder, and a cable driver locked loop are used when the transmitter does not send the clock on a separate line from the data (i.e. over telephone lines, or two wire cable). the NRZI encoder decoder is used in HDLC SDLC to combine the clock into the data line. A phase locked loop is used to recover the clock from the data line.

The majority of the available communication chips provide a limited number of data link control functions. Most of them will handle Zero Bit Insertion/Deletion (ZBI/D), Flags, Aborts, Automatic

230876-2

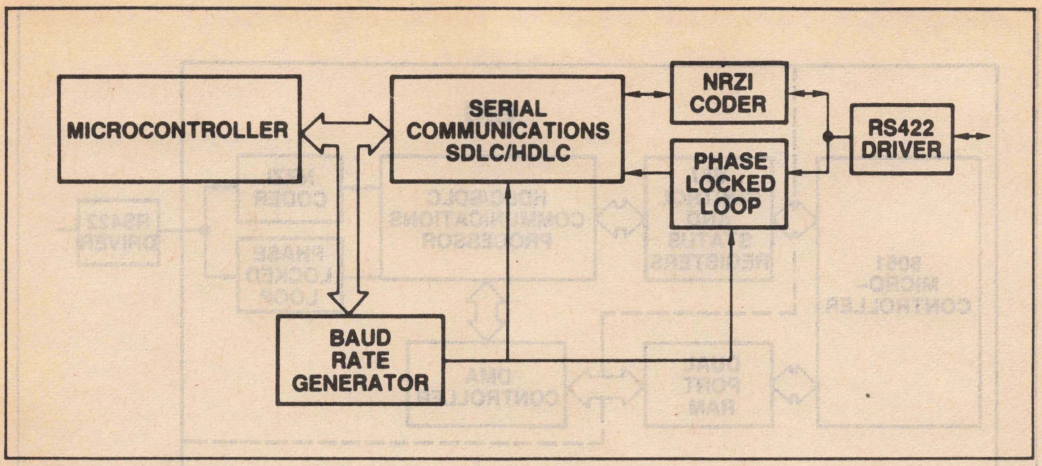


Figure 1. Conventional microcontroller networking solution

address recognition, and CRC generation and checking. It is the CPU's responsibility to manage link access, command recognition and response, acknowledgements and error recovery. Handling these tasks can take a lot of CPU time. In addition, servicing the transmission and reception of data bytes can also be very time consuming depending on the method used.

Using a DMA controller can increase the overall system performance, since it can transfer a block of data in fewer clock cycles than a CPU. In addition, the CPU and the DMA controller can multiplex their access to the bus so that both can be running at virtually the same time. However, both the DMA controller and the CPU are sharing the same bus, therefore, neither one get to utlize 100% of the bus bandwidth. Microcontrollers available today do not support DMA, therefore, they would have to use interrupts, since polling is unacceptable in a multitasking environment.

In an interrupt driven, the CPU has overhead in addition to servicing the interrupt. During each interrupt request the CPU has to save all of the important registers, transfer a byte, update pointers and counters, then restore all of its registers. At low bit rates this overhead may be insignificant. However, the percentage of overhead increases linearly with the bit rate. At high bit rates this overhead would consume all of the CPU's time. There is another nuisance factor associated with interrupt driven systems, interrupt latency. Too much interrupt latency will cause data to be lost from underrun and overrun errors.

The additional hardware necessary to implement the communications solution, as shown in Figure 1, would require 1 LSI chip and about 10 TTL chips. The cost of CPU throughput degradation can be even greater. The percentage of time the CPU has to spend servicing the communication tasks can be anywhere from 10-100%, depending on the serial bit rate. These high costs will prevent consumer acceptance of networking microcomputer equipment.

A Highly Integrated, High Performance Solution

The 8044 reduces the cost of networking microcontrollers without compromising performance. It contains all of the hardware components necessary to implement a microcomputer system with communications capability, plus it reduces the CPU and software overhead of implementing HDLC/SDLC. Figure 2 shows a functional block diagram of the 8044.

The 8044 integrates the powerful 8051 microcontroller with an intelligent Serial Interface Unit to provide a single chip solution which efficiently implements a distributed processing or distributed control system. The microcontroller is a self sufficient unit containing ROM, RAM, ALU and its own peripherals. The 8044's architecture and instruction set are identical to the 8051's. The Serial Interface Unit (SIU) uses bit synchronous HDLC/SDLC protocol and can communicate at bit rates up to 2.4 Mbps, externally clocked, or up to 375 Kbps using the on-chip digital phase locked loop. The SIU contains its own processor, which operates concurrently with the microcontroller.

The CPU and the SIU, in the 8044, interface through 192 bytes of dual port RAM. There is no hardware arbitration in the dual port RAM. Both processor's memory access cycles are interlaced; each processor has access every other clock cycle. Therefore, there

230876-3

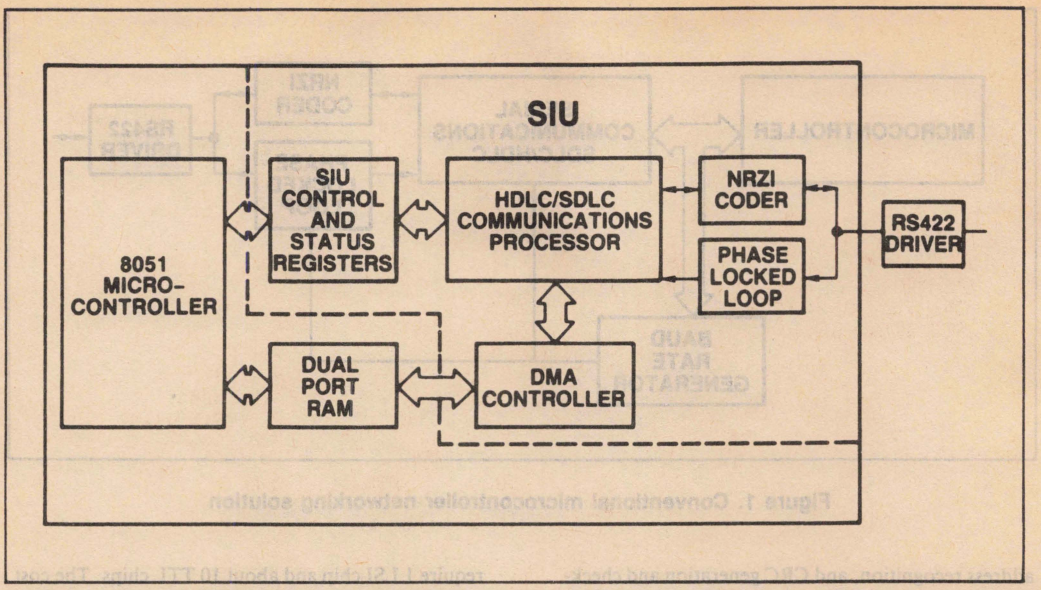


Figure 2. 8044 single chip microcontroller networking solution

is no throughput loss in either processor as a result of the dual port RAM, and execution times are deterministic. Since this has always been the method for memory access on the 8051 microcontroller, 8051 programs have the same execution time in the 8044.

By integrating all of the communication hardware onto the 8051 microcontroller, the hardware cost of the system is reduced. Now several chips have been integrated into a single chip. This means that the system power is reduced, P.C. board space is reduced, inventory and assembly is reduced, and reliability is improved. The improvement in reliability is a result of fewer chips and interconnections on the P.C. board.

As mentioned before, there can be two extremes in a design which adds communications to the microcomputer system. The 8044 solution uses the high end extreme. The SIU on the 8044 contains its own processor which communicates with the 8051 processor through dual port RAM and control/status registers. While the SIU is not a totally independent communications processor, it substantially offloads the 8051 processor from the communication tasks.

The DMA on the 8044 is dedicated to the SIU. It cannot access external RAM. By having a DMA controller in the SIU, the 8051 CPU is offloaded. As a result of the dual port RAM design, the DMA does not share the running at full speed while the frames are being transmitted or received. Also, the nuisance of overrun and underrun errors is totally eliminated since the dedicated DMA controller is guaranteed to meet the maximum data rates. Having a dedicated DMA controller means that the serial channel interrupt can be the lowest priority, thus allowing the CPU to have higher priority real time interrupts.

Figure 3 shows a comparison between the conventional and the 8044 solution on the percentage of time the CPU must spend sending data. This diagram was derived by assuming a 64 byte information frame is being transmitted repeatedly. The conventional solution is interrupt driven, and each interrupt service routine is assumed to take about 15 instructions with a 1 μsec instruction cycle time. At 533 Kbps, an interrupt would occur every 15 usec. Thus, the CPU becomes completely dedicated to servicing the serial communications. The conventional design could not support bit rates higher than this because of underruns and overruns. For the 8044 to repeatedly send 64 byte frames, it simply has to reinitialize the DMA controller. As a result, the 8044 can support bit rates up to 2.4 Mbps.

Some of the other communications tasks the CPU has to perform, such as link access, command recognition/response, and acknowledgements, are performed automatically by the SIU in a mode called "AUTO." The combination of the dedicated DMA controller and the AUTO mode, substantially offload

230876-4

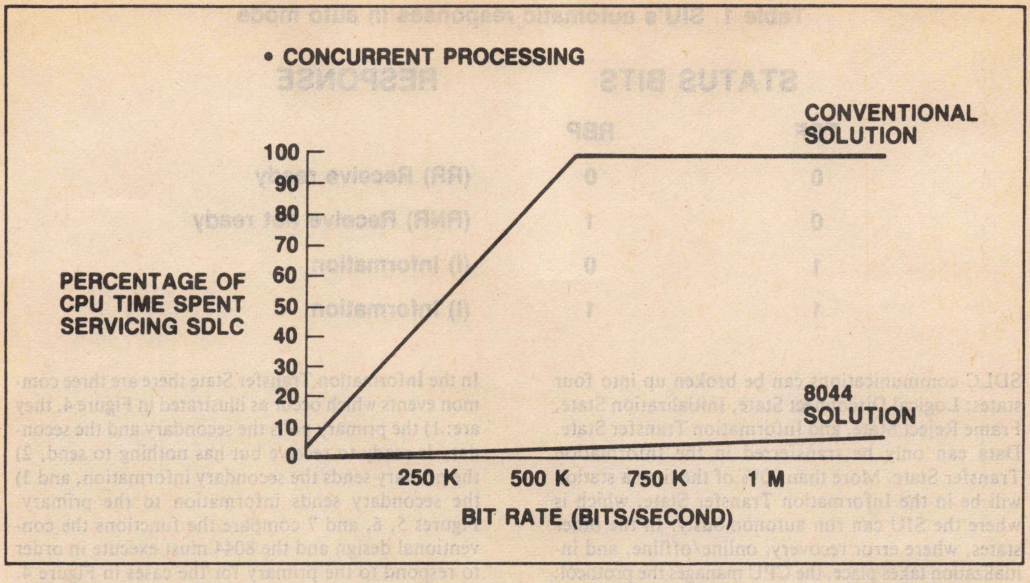


Figure 3. SIU offloads CPU

the CPU, thus allowing it to devote more of its power to other tasks.

8044's Auto Mode

In the AUTO mode the SIU implements in hardware a subset of the SDLC protocol such that it responds to many SDLC commands without CPU intervention. All AUTO mode responses to the primary station conform to IBM's SDLC definition. In the AUTO mode the 8044 can only be a secondary station operating in SDLC specified "Normal Response Mode." Normal Response Mode means that the secondary station can not transmit unless it is polled by the primary station. The SIU in the AUTO mode can recognize and respond to the following SDLC commands without CPU intervention: I (Information), RR (Receive Ready), RNR (Receive Not Ready), REJ (Reject), and for loop mode UP (Unnumbered Poll). The SIU can generate the following responses without CPU intervention: I, RR, and RNR. In addition, the SIU manages Ns and Nr in the control field. If it detects an error in either Ns or Nr, it interrupts the CPU for error recovery.

How does the SIU know what responses to send to the primary? It uses two status bits which are set by the CPU. The two bits are TBF (Transmit Buffer Full) and RBP (Receive Buffer Protect). TBF indicates that the CPU wants to send data, and RBP indicates that the receive data buffer is full. Table 1 shows the responses the SIU will send based on these two status bits. This is an innovative approach to communication design. The CPU in the 8044 with one instruction can directly set a bit which communicates to the primary what its transmit and receive buffering status is.

When the CPU wants to send a frame, it loads the transmit buffer with the data, loads the starting address and the count of the data into the SIU, then sets TBF to transmit the frame. The SIU waits for the primary station to poll it with a RR command. After the SIU is polled, it automatically sends the information frame to the primary with the proper control field. The SIU then waits for a positive acknowledgement from the primary before incrementing the Ns field and interrupting the CPU for more data. If a negative acknowledgement is received, the SIU automatically retransmits the frame.

When the 8044 is ready to receive information, the CPU loads the receive buffer starting address and the buffer length into the SIU, then enables the receiver. When a valid information frame with the correct address and CRC is received, the SIU will increment the Nr field, disable the receiver and interrupt the CPU indicating that a good I frame has been received. The CPU then sets RBP, reenables the receiver and processes the received data. By enabling the receiver with RBP set, the SIU will automatically respond to polls with a Receive Not Ready, thus keeping the link moving rather than timing out the primary from a disabled receiver, or interrupting the CPU with another poll before it has processed the data. After the data has been processed, the CPU clears RBP, returning to the Receive Ready responses.

230876-5

Table 1. SIU's automatic responses in auto mode

STATUS BITS		RESPONSE
TBF	RBP	
0	0	(RR) Receive ready
0	1	(RNR) Receive not ready
1	0	(I) Information
1	1	(I) Information

SDLC communications can be broken up into four states: Logical Disconnect State, Initialization State, Frame Reject State, and Information Transfer State. Data can only be transferred in the Information Transfer State. More than 90% of the time a station will be in the Information Transfer State, which is where the SIU can run autonomously. In the other states, where error recovery, online/offline, and initialization takes place, the CPU manages the protocol.

In the Information Transfer State there are three common events which occur as illustrated in Figure 4, they are: 1) the primary polls the secondary and the secondary is ready to receive but has nothing to send, 2) the primary sends the secondary information, and 3) the secondary sends information to the primary. Figures 5, 6, and 7 compare the functions the conventional design and the 8044 must execute in order to respond to the primary for the cases in Figure 4.

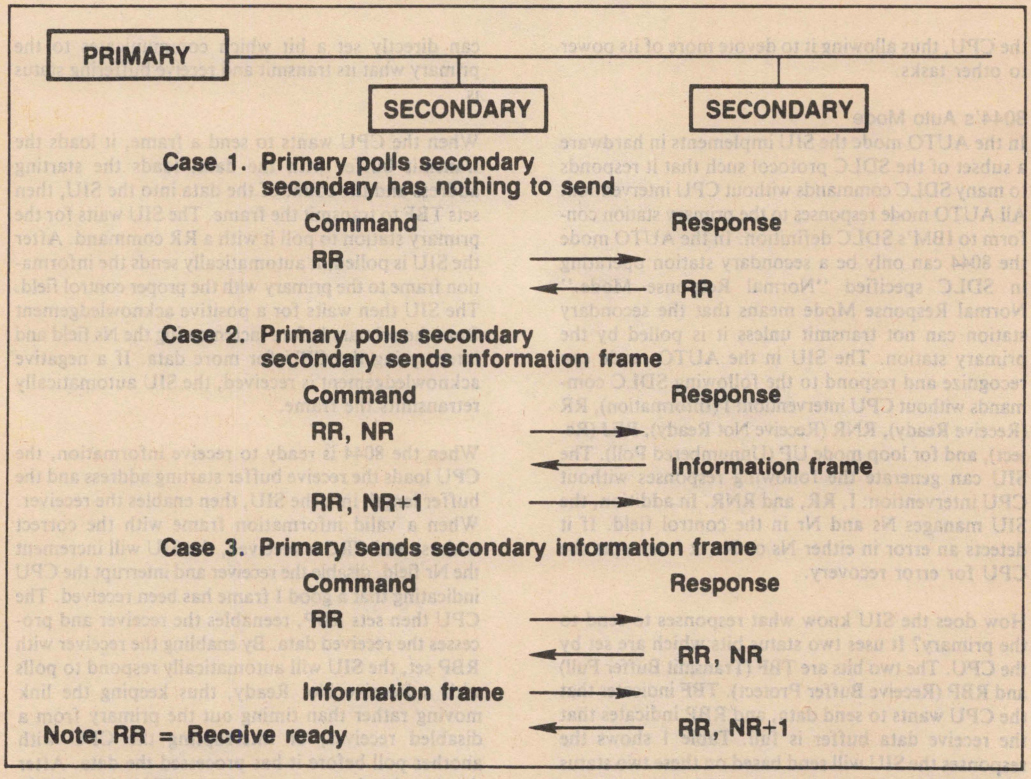


Figure 4. SDLC commands and responses in the information transfer state

230876-6

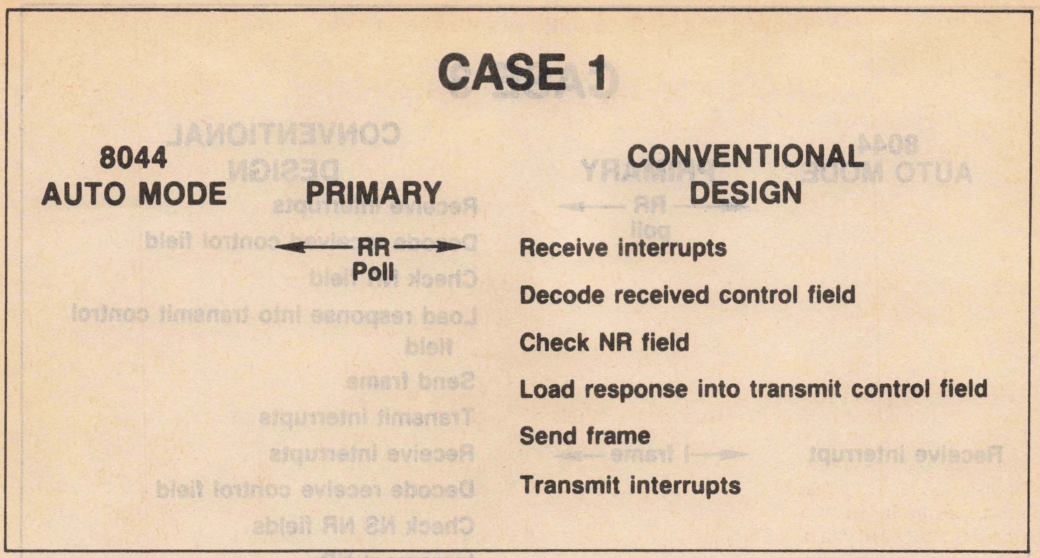


Figure 5. Primary polls secondary, secondary has nothing to send

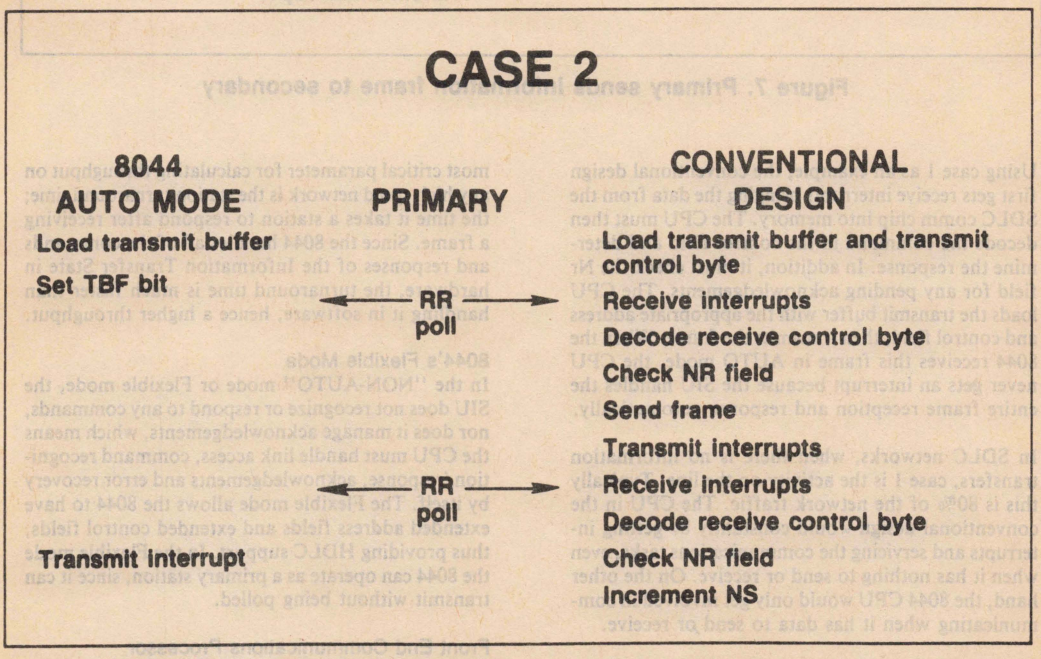


Figure 6. Primary polls secondary, secondary sends information frame

230876-7

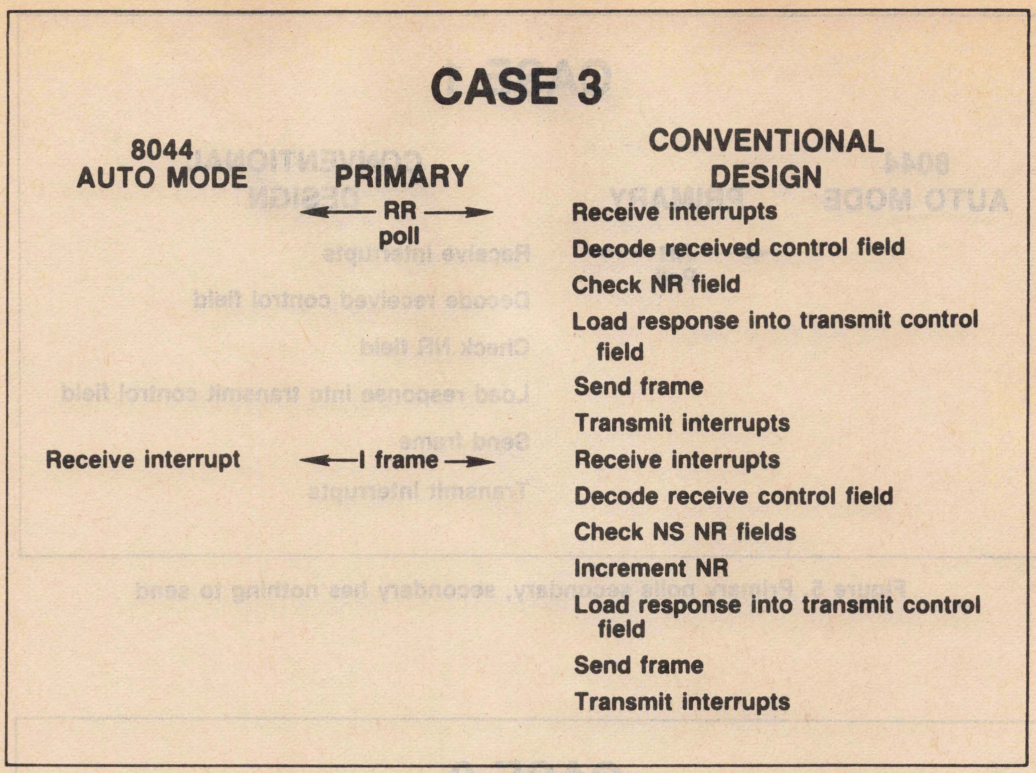


Figure 7. Primary sends information frame to secondary

Using case 1 as an example, the conventional design first gets receive interrupts bringing the data from the SDLC comm chip into memory. The CPU must then decode the command in the control field and determine the response. In addition, it must check the Nr field for any pending acknowledgements. The CPU loads the transmit buffer with the appropriate address and control field, then transmits the frame. When the 8044 receives this frame in AUTO mode, the CPU never gets an interrupt because the SIU handles the entire frame reception and response automatically.

In SDLC networks, when there is no information transfers, case 1 is the activity on the line. Typically this is 80% of the network traffic. The CPU in the conventional design would constantly be getting interrupts and servicing the communications tasks, even when it has nothing to send or receive. On the other hand, the 8044 CPU would only get involved in communicating when it has data to send or receive.

Having the SIU implement a subset of the SDLC protocol in hardware not only offloads the CPU, but it also improves the throughput on the network. The most critical parameter for calculating throughput on any high speed network is the station turnaround time; the time it takes a station to respond after receiving a frame. Since the 8044 handles all of the commands and responses of the Information Transfer State in hardware, the turnaround time is much faster than handling it in software, hence a higher throughput.

8044's Flexible Mode

In the "NON-AUTO" mode or Flexible mode, the SIU does not recognize or respond to any commands, nor does it manage acknowledgements, which means the CPU must handle link access, command recognition/response, acknowledgements and error recovery by itself. The Flexible mode allows the 8044 to have extended address fields and extended control fields, thus providing HDLC support. In the Flexible mode the 8044 can operate as a primary station, since it can transmit without being polled.

Front End Communications Processor

The 8044 can also be used as an intelligent HDLC/SDLC front end for a microporcessor, capable of extensively off-loading link control functions for

230876-8

the microporcessor. In some applications the 8044 can even be used for communications preprocessing, in addition to data link control. For this type of design the 8044 would communicate to the Host CPU through a FIFO, or dual port RAM. A block diagram of this design is given in Figure 8. A tightly coupled interface between the 8044 and the Host CPU would be established. The Host CPU would give the 8044 high level commands and data which the 8044 would convert to HDLC/SDLC. This particular type of design would be most appropriate for a primary Station which is normally a micro, mini, or mainframe computer. Sophisticated secondary stations could also take advantage of this design.

Since the 8044 has ROM on chip, all the communications software is non-volatile. The 8044 primary station could down-line-load software to 8044 secondary stations. Once down-line-loading is implemented, software updates to the primary and secondary stations could be done very inexpensively. The only things which would remain fixed in ROM are the HDLC/SDLC communications software and the software interface to the HOST.

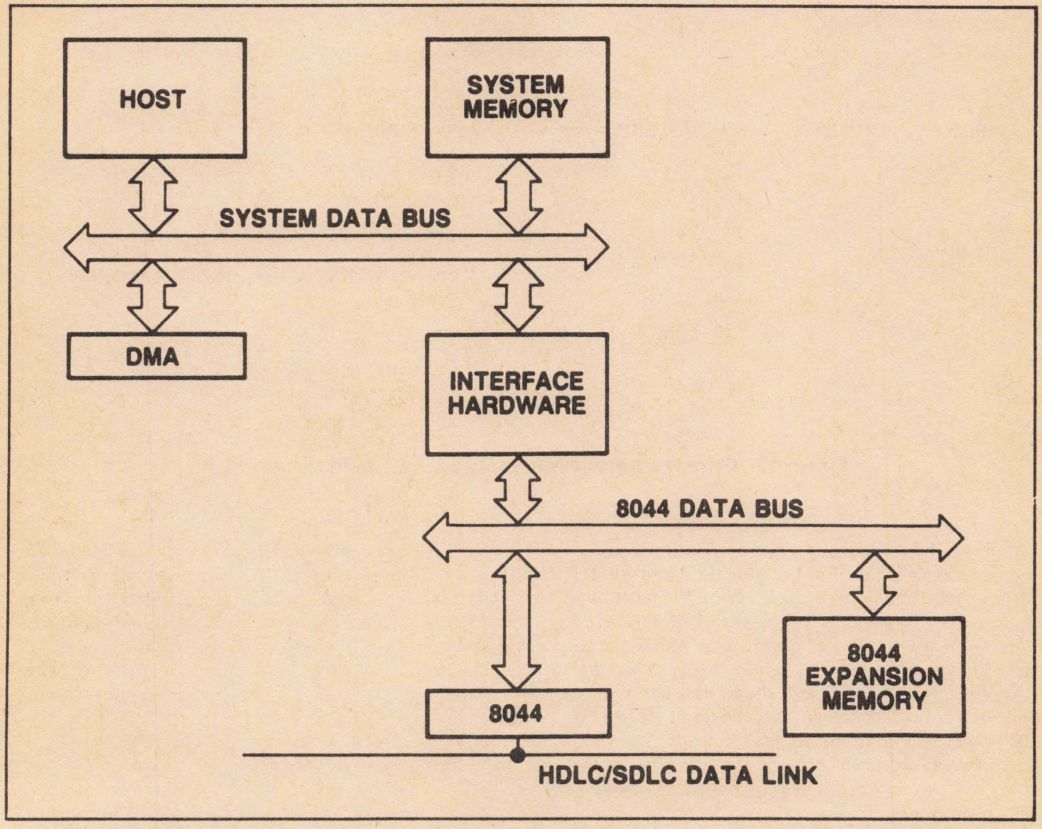
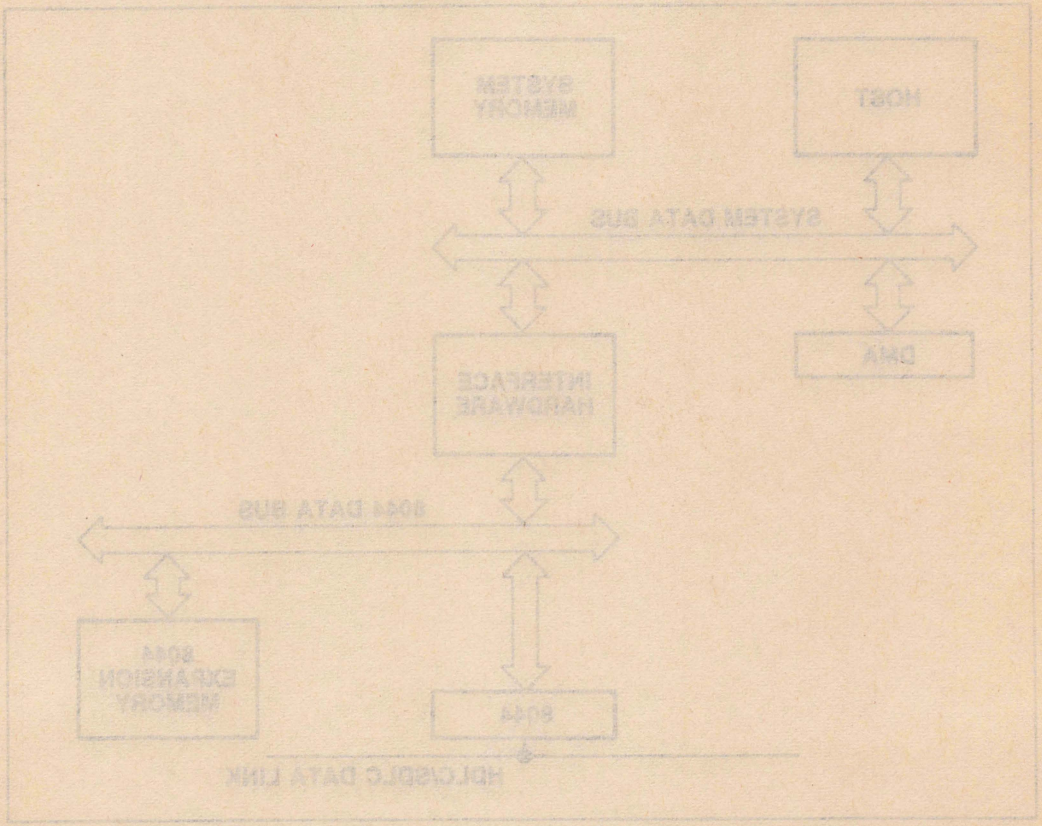


Figure 8. 8044 front end processor

230876-9


SYSTEM MEMORY
HOST
SYSTEM DATA BUS
INTERFACE HARDWARE
DMA
8044 DATA BUS
8044 EXPANSION MEMORY
8044
HDLC/SDLC DATA LINK

Design Consideration Application Notes

25

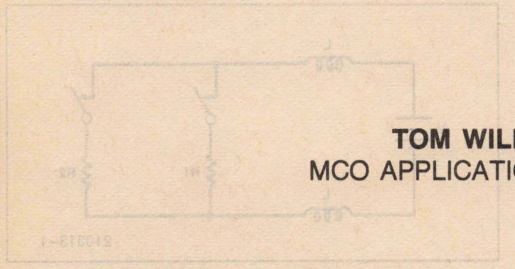
intel

APPLICATION NOTE

AP-125

November 1986

Designing Microcontroller Systems for Electrically Noisy Environments



TOM WILLIAMSON
MCO APPLICATIONS ENGINEER

Order Number: 210313-002

Digital circuits are often thought of as being immune to noise problems, but really they're not. Noises in digital systems produce software upsets: program jumps to apparently random locations in memory. Noise-induced glitches in the signal lines can cause such problems, but the supply voltage is more sensitive to glitches than the signal lines.

Severe noise conditions, those involving electrostatic discharges, or as found in automotive environments, can do permanent damage to the hardware. Electrostatic discharges can blow a crater in the silicon. In the automotive environment, in ordinary operation, the "12V" power line can shown + and −400V transients.

This Application Note describes some electrical noises and noise environments. Design considerations, along the lines of PCB layout, power supply distribution and decoupling, and shielding and grounding techniques, that may help minimize noise susceptibility are reviewed. Special attention is given to the automotive and ESD environments.

Symptoms of Noise Problems

Noise problems are not usually encountered during the development phase of a microcontroller system. This is because benches rarely simulate the system's intended environment. Noise problems tend not to show up until the system is installed and operating in its intended environment. Then, after a few minutes or hours of normal operation the system finds itself someplace out in left field. Inputs are ignored and outputs are gibberish. The system may respond to a reset, or it may have to be turned off physically and then back on again, at which point it commences operating as though nothing had happened. There may be an obvious cause, such as an electrostatic discharge from somebody's finger to a keyboard or the upset occurs every time a copier machine is turned on or off. Or there may be no obvious cause, and nothing the operator can do will make the upset repeat itself. But a few minutes, or a few hours, or a few days later it happens again.

One symptom of electrical noise problems is randomness, both in the occurrence of the problem and in what the system does in its failure. All operational upsets that occur at seemingly random intervals are not necessarily caused by noise in the system. Marginal VCC, inadequate decoupling, rarely encountered software conditions, or timing coincidences can produce upsets that seem to occur randomly. On the other hand, some noise sources can produce upsets downright periodically. Nevertheless, the more difficult it is to characterize an upset as to cause and effect, the more likely it is to be a noise problem.

Types and Sources of Electrical Noise

The name given to electrical noises other than those that are inherent in the circuit components (such as thermal noise) is EMI: electromagnetic interference. Motors, power switches, fluorescent lights, electrostatic discharges, etc., are sources of EMI. There is a veritable alphabet soup of EMI types, and these are briefly described below.

SUPPLY LINE TRANSIENTS

Anything that switches heavy current loads onto or off of AC or DC power lines will cause large transients in these power lines. Switching an electric typewriter on or off, for example, can put a 1000V spike onto the AC power lines.

The basic mechanism behind supply line transients is shown in Figure 1. The battery represents any power source, AC or DC. The coils represent the line inductance between the power source and the switchable loads R1 and R2. If both loads are drawing current, the line current flowing through the line inductance establishes a magnetic field of some value. Then, when one of the loads is switched off, the field due to that component of the line current collapses, generating transient voltages, $v = L(di/dt)$, which try to maintain the current at its original level. That's called an "inductive kick." Because of contact bounce, transients are generated whether the switch is being opened or closed, but they're worse when the switch is being opened.

An inductive kick of one type or another is involved in most line transients, including those found in the automotive environment. Other mechanisms for line transients exist, involving noise pickup on the lines. The noise voltages are then conducted to a susceptible circuit right along with the power.

EMP AND RFI

Anything that produces arcs or sparks will radiate electromagnetic pulses (EMP) or radio-frequency interference (RFI).

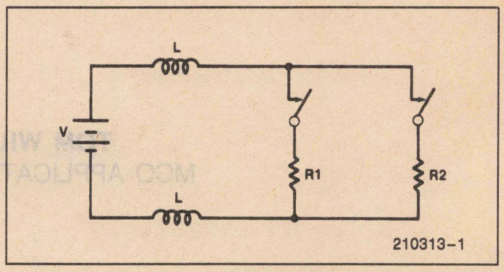


Figure 1. Supply Line Transients

Spark discharges have probably caused more software upsets in digital equipment than any other single noise source. The upsetting mechanism is the EMP produced by the spark. The EMP induces transients in the circuit, which are what actually cause the upset.

Arcs and sparks occur in automotive ignition systems, electric motors, switches, static discharges, etc. Electric motors that have commutator bars produce an arc as the brushes pass from one bar to the next. DC motors and the "universal" (AC/DC) motors that are used to power hand tools are the kinds that have commutator bars. In switches, the same inductive kick that puts transients on the supply lines will cause an opening or closing switch to throw a spark.

ESD

Electrostatic discharge (ESD) is the spark that occurs when a person picks up a static charge from walking across a carpet, and then discharges it into a keyboard, or whatever else can be touched. Walking across a carpet in a dry climate, a person can accumulate a static voltage of 35kV. The current pulse from an electrostatic discharge has an extremely fast risetime — typically, 4A/ns. Figure 2 shows ESD waveforms that have been observed by some investigators of ESD phenomena.

It is enlightening to calculate the L(di/dt) voltage required to drive an ESD current pulse through a couple of inches of straight wire. Two inches of straight wire has about 50 nH of inductance. That's not very much, but using 50 nH for L and 4A/ns for di/dt gives an L(di/dt) drop of about 200V. Recent observations by W.M. King suggest even faster risetimes (Figure 2b) and the occurrence of multiple discharges during a single discharge event.

Obviously, ESD-sensitivity needs to be considered in the design of equipment that is going to be subjected to it, such as office equipment.

GROUND NOISE

Currents in ground lines are another source of noise. These can be 60 Hz currents from the power lines, or RF hash, or crosstalk from other signals that are sharing this particular wire as a signal return line. Noise in the ground lines is often referred to as a "ground loop" problem. The basic concept of the ground loop is shown in Figure 3. The problem is that true earth-ground is not really at the same potential in all locations. If the two ends of a wire are earth-grounded at different locations, the voltage difference between the two "ground" points can drive significant currents (several amperes) through the wire. Consider the wire to be part of a loop which contains, in addition to the wire, a voltage source that represents the difference in potential between the two ground points, and you have the classical "ground loop." By extension, the term is used to refer to any unwanted (and often unexpected) currents in a ground line.

"Radiated" and "Conducted" Noise

Radiated noise is noise that arrives at the victim circuit in the form of electromagnetic radiation, such as EMP and RFI. It causes trouble by inducing extraneous voltages in the circuit. Conducted noise is noise that arrives at the victim circuit already in the form of an extraneous voltage, typically via the AC or DC power lines.

One defends against radiated noise by care in designing layouts and the use of effective shielding techniques. One defends against conducted noise with filters and

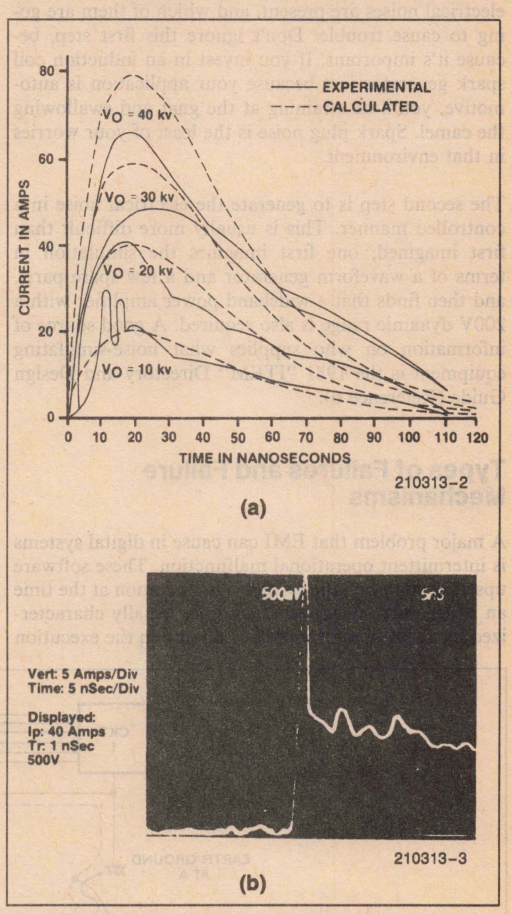


Figure 2. Waveforms of Electrostatic Discharge Currents From a Hand-Held Metallic Object

suppressors, although layouts and grounding techniques are important here, too.

Simulating the Environment

Addressing noise problems after the design of a system has been completed is an expensive proposition. The ill will generated by failures in the field is not cheap either. It's cheaper in the long run to invest a little time and money in learning about noise and noise simulation equipment, so that controlled tests can be made on the bench as the design is developing.

Simulating the intended noise environment is a two-step process: First you have to recognize what the noise environment is, that is, you have to know what kinds of electrical noises are present, and which of them are going to cause trouble. Don't ignore this first step, because it's important. If you invest in an induction coil spark generator just because your application is automotive, you'll be straining at the gnat and swallowing the camel. Spark plug noise is the least of your worries in that environment.

The second step is to generate the electrical noise in a controlled manner. This is usually more difficult than first imagined; one first imagines the simulation in terms of a waveform generator and a few spare parts, and then finds that a wideband power amplifier with a 200V dynamic range is also required. A good source of information on who supplies what noise-simulating equipment is the 1981 "ITEM" Directory and Design Guide (Reference 6).

Types of Failures and Failure Mechanisms

A major problem that EMI can cause in digital systems is intermittent operational malfunction. These software upsets occur when the system is in operation at the time an EMI source is activated, and are usually characterized by a loss of information or a jump in the execution of the program to some random location in memory. The person who has to iron out such problems is tempted to say the program counter went crazy. There is usually no damage to the hardware, and normal operation can resume as soon as the EMI has passed or the source is de-activated. Resuming normal operation usually requires manual or automatic reset, and possibly re-entering of lost information.

Electrostatic discharges from operating personnel can cause not only software upsets, but also permanent ("hard") damage to the system. For this to happen the system doesn't even have to be in operation. Sometimes the permanent damage is latent, meaning the initial damage may be marginal and require further aggravation through operating stress and time before permanent failure takes place. Sometimes too the damage is hidden.

One ESD-related failure mechanism that has been identified has to do with the bias voltage on the substrate of the chip. On some CPU chips the substrate is held at −2.5V by a phase-shift oscillator working into a capacitor/diode clamping circuit. This is called a "charge pump" in chip-design circles. If the substrate wanders too far in either direction, program read errors are noted. Some designs have been known to allow electrostatic discharge currents to flow directly into port pins of an 8048. The resulting damage to the oxide causes an increase in leakage current, which loads down the charge pump, reducing the substrate voltage to a marginal or unacceptable level. The system is then unreliable or completely inoperative until the CPU chip is replaced. But if the CPU chip was subjected to a discharge spark once, it will eventually happen again.

Chips that have a grounded substrate, such as the 8748, can sometimes sustain some oxide damage without actually becoming inoperative. In this case the damage is present, and the increased leakage current is noted; however, since the substrate voltage retains its design value, the damage is largely hidden.

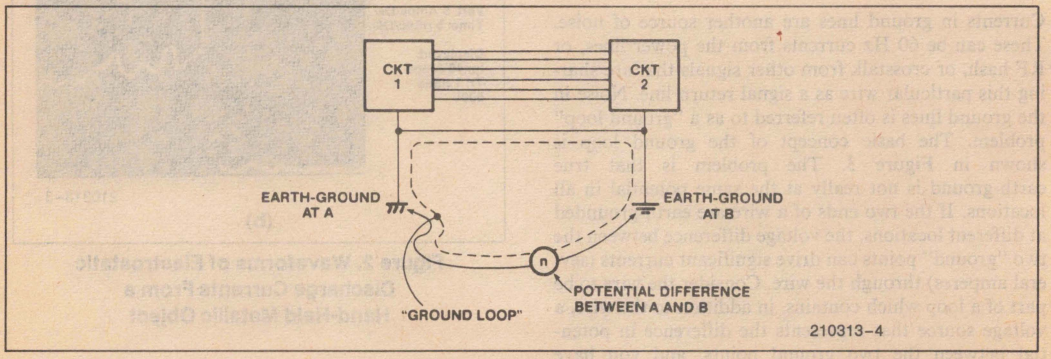


Figure 3. What a Ground Loop Is

It must therefore be recognized that connecting port pins unprotected to a keyboard or to anything else that is subject to electrostatic discharges, makes an extremely dangerous configuration. It doesn't make any difference what CPU chip is being used, or who makes it. If it connects unprotected to a keyboard, it will eventually be destroyed. Designing for an ESD-environment will be discussed further on.

We might note here that MOS chips are not the only components that are susceptible to permanent ESD damage. Bipolar and linear chips can also be damaged in this way. PN junctions are subject to a hard failure mechanism called thermal secondary breakdown, in which a current spike, such as from an electrostatic discharge, causes microscopically localized spots in the junction to approach melt temperatures. Low power TTL chips are subject to this type of damage, as are op-amps. Op-amps, in addition, often carry on-chip MOS capacitors which are directly across an external pin combination, and these are susceptible to dielectric breakdown.

We return now to the subject of software upsets. Noise transients can upset the chip through any pin, even an output pin, because every pin on the chip connects to the substrate through a pn junction. However, the most vulnerable pin is probably the VCC line, since it has direct access to all parts of the chip: every register, gate, flip-flop and buffer.

The menu of possible upset mechanisms is quite lengthy. A transient on the substrate at the wrong time will generally cause a program read error. A false level at a control input can cause an extraneous or misdirected opcode fetch. A disturbance on the supply line can flip a bit in the program counter or instruction register. A short interruption or reversal of polarity on the supply line can actually turn the processor off, but not long enough for the power-up reset capacitor to discharge. Thus when the transient ends, the chip starts up again without a reset.

A common failure mode is for the processor to lock itself into a tight loop. Here it may be executing the data in a table, or the program counter may have jumped a notch, so that the processor is now executing operands instead of opcodes, or it may be trying to fetch opcodes from a nonexistent external program memory.

It should be emphasized that mechanisms for upsets have to do with the arrival of noise-induced transients at the pins of the chips, rather than with the generation of noise pulses within the chip itself, that is, it's not the chip that is picking up noise, it's the circuit.

The Game Plan

Prevention is usually cheaper than suppression, so first we'll consider some preventive methods that might help to minimize the generation of noise voltages in the circuit. These methods involve grounding, shielding, and wiring techniques that are directed toward the mechanisms by which noise voltages are generated in the circuit. We'll also discuss methods of decoupling. Then we'll look at some schemes for making a graceful recovery from upsets that occur in spite of preventive measures. Lastly, we'll take another look at two special problem areas: electrostatic discharges and the automotive environment.

Current Loops

The first thing most people learn about electricity is that current won't flow unless it can flow in a closed loop. This simple fact is sometimes temporarily forgotten by the overworked engineer who has spent the past several years mastering the intricacies of the DO loop, the timing loop, the feedback loop, and maybe even the ground loop. The simple current loop probably owes its apparent demise to the invention of the ground symbol. By a stroke of the pen one avoids having to draw the return paths of most of the current loops in the circuit. Then "ground" turns into an infinite current sink, so that any current that flows into it is gone and forgotten. Forgotten it may be, but it's not gone. It must return to its source, so that its path will by all the laws of nature form a closed loop.

The physical geometry of a given current loop is the key to why it generates EMI, why it's susceptible to EMI, and how to shield it. Specifically, it's the area of the loop that matters.

Any flow of current generates a magnetic field whose intensity varies inversely to the distance from the wire that carries the current. Two parallel wires conducting currents $+I$ and $-I$ (as in signal feed and return lines) would generate a nonzero magnetic field near the wires, where the distance from a given point to one wire is noticeably different from the distance to the other wire, but farther away (relative to the wire spacing), where the distances from a given point to either wire are about the same, the fields from both wires tend to cancel out. Thus, maintaining proximity between feed and return paths is an important way to minimize their interference with other signals. The way to maintain their proximity is essentially to minimize their loop area. And, because the mutual inductance from current loop A to current loop B is the same as the mutual inductance from current loop B to current loop A, a circuit that doesn't radiate interference doesn't receive it either.

Thus, from the standpoint of reducing both generation of EMI and susceptibility to EMI, the hard rule is to keep loop areas small. To say that loop areas should be minimized is the same as saying the circuit inductance

should be minimized. Inductance is by definition the constant of proportionality between current and the magnetic field it produces: $\phi = LI$. Holding the feed and return wires close together so as to promote field cancellation can be described either as minimizing the loop area or as minimizing L. It's the same thing.

Shielding

There are three basic kinds of shields: shielding against capacitive coupling, shielding against inductive coupling, and RF shielding. Capacitive coupling is electric field coupling, so shielding against it amounts to shielding against electric fields. As will be seen, this is relatively easy. Inductive coupling is magnetic field coupling, so shielding against it is shielding against magnetic fields. This is a little more difficult. Strangely enough, this type of shielding does not in general involve the use of magnetic materials. RF shielding, the classical "metallic barrier" against all sorts of electromagnetic fields, is what most people picture when they think about shielding. Its effectiveness depends partly on the selection of the shielding material, but mostly, as it turns out, on the treatment of its seams and the geometry of its openings.

SHIELDING AGAINST CAPACITIVE COUPLING

Capacitive coupling involves the passage of interfering signals through mutual or stray capacitances that aren't shown on the circuit diagram, but which the experienced engineer knows are there. Capacitive coupling to one's body is what would cause an unstable oscillator to change its frequency when the person reaches his hand over the circuit, for example. More importantly, in a digital system it causes crosstalk in multi-wire cables.

The way to block capacitive coupling is to enclose the circuit or conductor you want to protect in a metal shield. That's called an electrostatic or Faraday shield. If coverage is 100%, the shield does not have to be grounded, but it usually is, to ensure that circuit-to-shield capacitances go to signal reference ground rather than act as feedback and crosstalk elements. Besides, from a mechanical point of view, grounding it is almost inevitable.

A grounded Faraday shield can be used to break capacitive coupling between a noisy circuit and a victim circuit, as shown in Figure 4. Figure 4a shows two circuits capacitively coupled through the stray capacitance between them. In Figure 4b the stray capacitance is intercepted by a grounded Faraday shield, so that interference currents are shunted to ground. For example, a grounded plane can be inserted between PCBs (printed circuit boards) to eliminate most of the capacitive coupling between them.

Another application of the Faraday shield is in the electrostatically shielded transformer. Here, a conducting foil is laid between the primary and secondary coils so as to intercept the capacitive coupling between them. If a system is being upset by AC line transients, this type of transformer may provide the fix. To be effective in this application, the shield must be connected to the greenwire ground.

SHIELDING AGAINST INDUCTIVE COUPLING

With inductive coupling, the physical mechanism involved is a magnetic flux density B from some external interference source that links with a current loop in the victim circuit, and generates a voltage in the loop in accordance with Lenz's law: $v = -NA(dB/dt)$, where in this case $N = 1$ and A is the area of the current loop in the victim circuit.

There are two aspects to defending a circuit against inductive pickup. One aspect is to try to minimize the offensive fields at their source. This is done by minimizing the area of the current loop at the source so as to promote field cancellation, as described in the section on current loops. The other aspect is to minimize the inductive pickup in the victim circuit by minimizing the area of that current loop, since, from Lenz's law, the induced voltage is proportional to this area. So the two aspects really involve the same corrective action: minimize the areas of the current loops. In other words, minimizing the offensiveness of a circuit inherently minimizes its susceptibility.

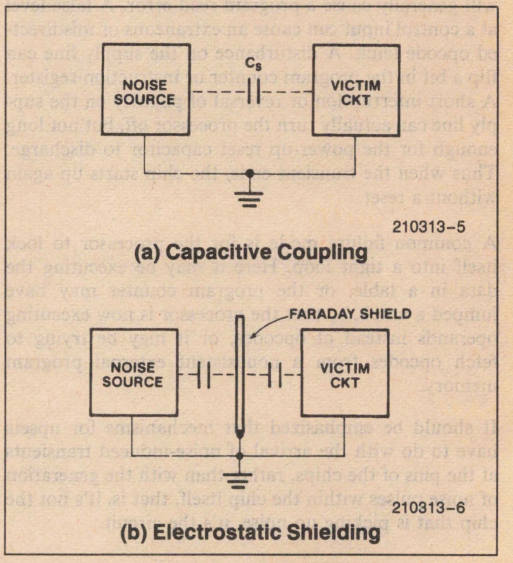


Figure 4. Use of Faraday Shield

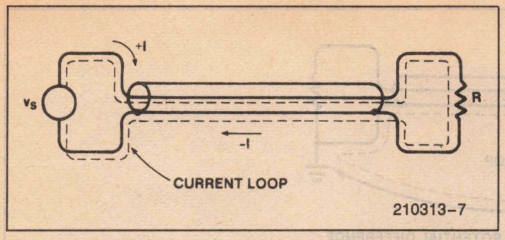


Figure 5. External to the Shield, $\phi = 0$

Shielding against inductive coupling means nothing more nor less than controlling the dimensions of the current loops in the circuit. We must look at four examples of this type of "shielding": the coaxial cable, the twisted pair, the ground plane, and the gridded-ground PCB layout.

The Coaxial Cable—Figure 5 shows a coaxial cable carrying a current I from a signal source to a receiving load. The shield carries the same current as the center conductor. Outside the shield, the magnetic field produced by +I flowing in the center conductor is cancelled by the field produced by −I flowing in the shield. To the extent that the cable is ideal in producing zero external magnetic field, it is immune to inductive pickup from external sources. The cable adds effectively zero area to the loop. This is true only if the shield carries the same current as the center conductor.

In the real world, both the signal source and the receiving load are likely to have one end connected to a common signal ground. In that case, should the cable be grounded at one end, both ends, or neither end? The answer is that it should be grounded at both ends. Figure 6a shows the situation when the cable shield is grounded at only one end. In that case the current loop runs down the center conductor of the cable, then back through the common ground connection. The loop area is not well defined. The shield not only does not carry the same current as the center conductor, but it doesn't carry any current at all. There is no field cancellation at all. The shield has no effect whatsoever on either the generation of EMI or susceptibility to EMI. (It is, however, still effective as an electrostatic shield, or at least it would be if the shield coverage were 100%.)

Figure 6b shows the situation when the cable is grounded at both ends. Does the shield carry all of the return current, or only a portion of it on account of the shunting effect of the common ground connection? The answer to that question depends on the frequency content of the signal. In general, the current loop will follow the path of least impedance. At low frequencies, 0 Hz to several kHz, where the inductive reactance is insignificant, the current will follow the path of least resistance. Above a few kHz, where inductive reactance predominates, the current will follow the path of least inductance. The path of least inductance is the path of

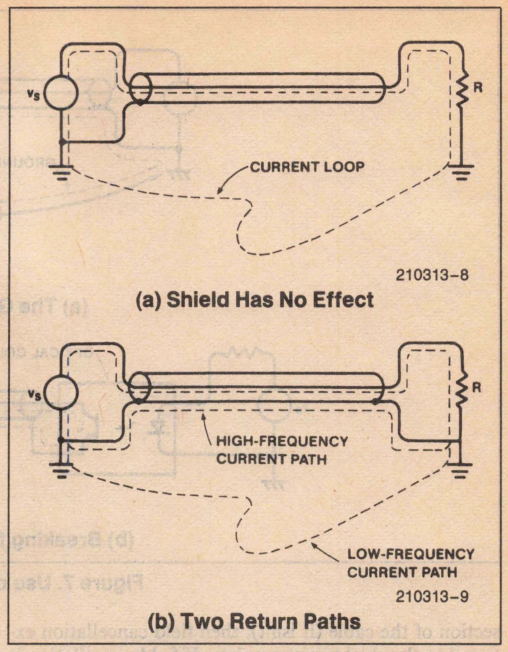


Figure 6. Use of Coaxial Cable

minimum loop area. Hence, for higher frequencies the shield carries virtually the same current as the center conductor, and is therefore effective against both generation and reception of EMI.

Note that we have now introduced the famous "ground loop" problem, as shown in Figure 7a. Fortunately, a digital system has some built-in immunity to moderate ground loop noise. In a noisy environment, however, one can break the ground loop, and still maintain the shielding effectiveness of the coaxial cable, by inserting an optical coupler, as shown in Figure 7b. What the optical coupler does, basically, is allow us to re-define the signal source as being ungrounded, so that that end of the cable need not be grounded, and still lets the shield carry the same current as the center conductor. Obviously, if the signal source weren't grounded in the first place, the optical coupler wouldn't be needed.

The Twisted Pair—A cheaper way to minimize loop area is to run the feed and return wires right next to each other. This isn't as effective as a coaxial cable in minimizing loop area. An ideal coaxial cable adds zero area to the loop, whereas merely keeping the feed and return wires next to each other is bound to add a finite area.

However, two things work to make this cheaper method almost as good as a coaxial cable. First, real coaxial cables are not ideal. If the shield current isn't evenly distributed around the center conductor at every cross-

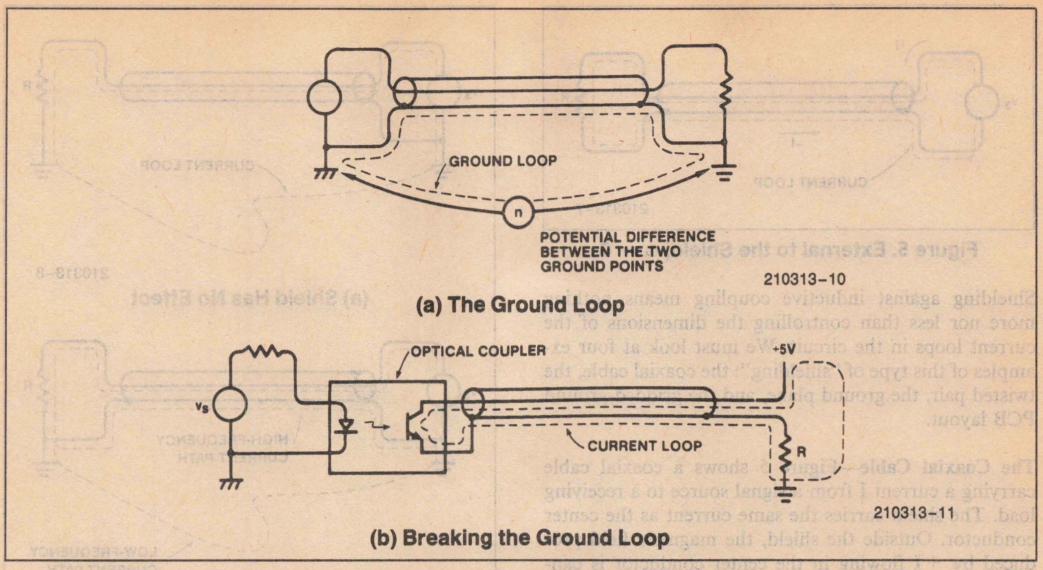


Figure 7. Use of Optical Coupler

section of the cable (it isn't), then field cancellation external to the shield is incomplete. If field cancellation is incomplete, then the effective area added to the loop by the cable isn't zero. Second, in the cheaper method the feed and return wires can be twisted together. This not only maintains their proximity, but the noise picked up in one twist tends to cancel out the noise picked up in the next twist down the line. Thus the "twisted pair" turns out to be about as good a shield against inductive coupling as coaxial cable is.

The twisted pair does not, however, provide electrostatic shielding (i.e., shielding against capacitive coupling). Another operational difference between them is that the coaxial cable works better at higher frequencies. This is primarily because the twisted pair adds more capacitive loading to the signal source than the coaxial cable does. The twisted pair is normally considered useful up to only about 1 MHz, as opposed to near a GHz for the coaxial cable.

The Ground Plane—The best way to minimize loop areas when many current loops are involved is to use a ground plane. A ground plane is a conducting surface that is to serve as a return conductor for all the current loops in the circuit. Normally, it would be one or more layers of a multilayer PCB. All ground points in the circuit go not to a grounded trace on the PCB, but directly to the ground plane. This leaves each current loop in the circuit free to complete itself in whatever configuration yields minimum loop area (for frequencies wherein the ground path impedance is primarily inductive).

Thus, if the feed path for a given signal zigzags its way across the PCB, the return path for this signal is free to zigzag right along beneath it on the ground plane, in such a configuration as to minimize the energy stored in the magnetic field produced by this current loop. Minimal magnetic flux means minimal effective loop area and minimal susceptibility to inductive coupling.

The Gridded-Ground PCB Layout—The next best thing to a ground plane is to lay out the ground traces on a PCB in the form of a grid structure, as shown in Figure 8. Laying horizontal traces on one side of the board and vertical traces on the other side allows the passage of signal and power traces. Wherever vertical and horizontal ground traces cross, they must be connected by a feed-through.

Have we not created here a network of "ground loops"? Yes, in the literal sense of the word, but loops in the ground layout on a PCB are not to be feared. Such inoffensive little loops have never caused as much noise pickup as their avoidance has. Trying to avoid innocent little loops in the ground layout, PCB designers have forced current loops into geometries that could swallow a whale. That is exactly the wrong thing to do.

The gridded ground structure works almost as well as the ground plane, as far as minimizing loop area is concerned. For a given current loop, the primary return path may have to zig once in a while where its feed path zags, but you still get a mathematically optimal dis-

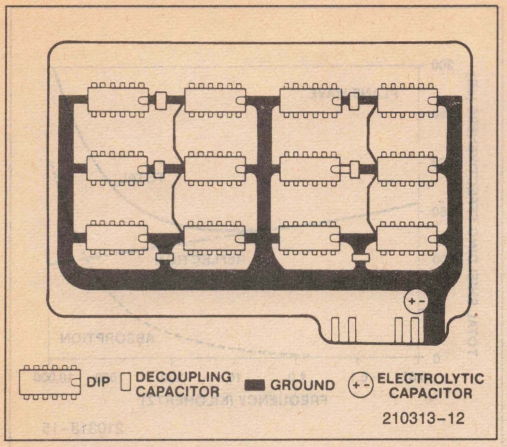


Figure 8. PCB with Gridded Ground

tribution of currents in the grid structure, such that the current loop produces less magnetic flux than if the return path were restrained to follow any single given ground trace. The key to attaining minimum loop areas for all the current loops together is to let the ground currents distribute themselves around the entire area of the board as freely as possible. They want to minimize their own magnetic field. Just let them.

RF SHIELDING

A time-varying electric field generates a time-varying magnetic field, and vice versa. Far from the source of a time-varying EM field, the ratio of the amplitudes of the electric and magnetic fields is always 377Ω. Up close to the source of the fields, however, this ratio can be quite different, and dependent on the nature of the source. Where the ratio is near 377Ω is called the far field, and where the ratio is significantly different from 377Ω is called the near field. The ratio itself is called the wave impedance, E/H.

The near field goes out about 1/6 of a wavelength from the source. At 1 MHz this is about 150 feet, and at 10 MHz it's about 15 feet. That means if an EMI source is in the same room with the victim circuit, it's likely to be a near field problem. The reason this matters is that in the near field an RF interference problem could be almost entirely due to E-field coupling or H-field coupling, and that could influence the choice of an RF shield or whether an RF shield will help at all.

In the near field of a whip antenna, the E/H ratio is higher than 377Ω, which means it's mainly an E-field generator. A wire-wrap post can be a whip antenna. Interference from a whip antenna would be by electric field coupling, which is basically capacitive coupling. Methods to protect a circuit from capacitive coupling, such as a Faraday shield, would be effective against RF interference from a whip antenna. A gridded-ground structure would be less effective.

In the near field of a loop antenna, the E/H ratio is lower than 377Ω, which means it's mainly an H-field generator. Any current loop is a loop antenna. Interference from a loop antenna would be by magnetic field coupling, which is basically the same as inductive coupling. Methods to protect a circuit from inductive coupling, such as a gridded-ground structure, would be effective against RF interference from a loop antenna. A Faraday shield would be less effective.

A more difficult case of RF interference, near field or far field, may require a genuine metallic RF shield. The idea behind RF shielding is that time-varying EMI fields induce currents in the shielding material. The induced currents dissipate energy in two ways: I^2R losses in the shielding material and radiation losses as they re-radiate their own EM fields. The energy for both of these mechanisms is drawn from the impinging EMI fields. Hence the EMI is weakened as it penetrates the shield.

More formally, the I^2R losses are referred to as absorption loss, and the re-radiation is called reflection loss. As it turns out, absorption loss is the primary shielding mechanism for H-fields, and reflection loss is the primary shielding mechanism for E-fields. Reflection loss, being a surface phenomenon, is pretty much independent of the thickness of the shielding material. Both loss mechanisms, however, are dependent on the frequency (ω) of the impinging EMI field, and on the permeability (μ) and conductivity (σ) of the shielding material. These loss mechanisms vary approximately as follows:

$$\text{reflection loss to an E-field (in dB)} \sim \log \frac{\sigma}{\omega\mu}$$

$$\text{absorption loss to an H-field (in dB)} \sim t\sqrt{\omega\sigma\mu}$$

where t is the thickness of the shielding material.

The first expression indicates that E-field shielding is more effective if the shield material is highly conductive, and less effective if the shield if ferromagnetic, and that low-frequency fields are easier to block than high-frequency fields. This is shown in Figure 9.

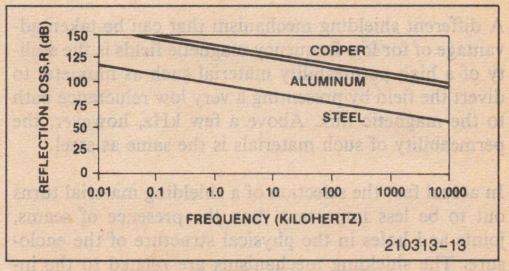


Figure 9. E-Field Shielding

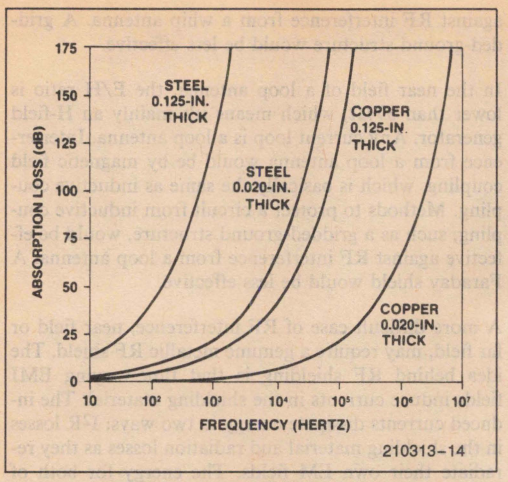


Figure 10. H-Field Shielding

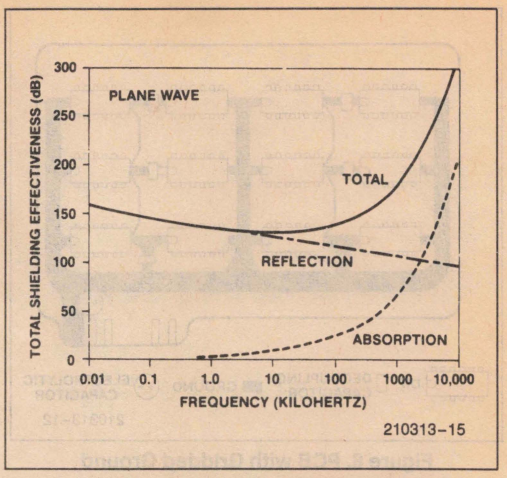


Figure 11. E- and H-Field Shielding

Copper and aluminum both have the same permeability, but copper is slightly more conductive, and so provides slightly greater reflection loss to an E-field. Steel is less effective for two reasons. First, it has a somewhat elevated permeability due to its iron content, and second, as tends to be the case with magnetic materials, it is less conductive.

On the other hand, according to the expression for absorption loss to an H-field, H-field shielding is more effective at higher frequencies and with shield material that has both high conductivity and high permeability. In practice, however, selecting steel for its high permeability involves some compromise in conductivity. But the increase in permeability more than makes up for the decrease in conductivity, as can be seen in Figure 10. This figure also shows the effect of shield thickness.

A composite of E-field and H-field shielding is shown in Figure 11. However, this type of data is meaningful only in the far field. In the near field the EMI could be 90% H-field, in which case the reflection loss is irrelevant. It would be advisable then to beef up the absorption loss, at the expense of reflection loss, by choosing steel. A better conductor than steel might be less expensive, but quite ineffective.

A different shielding mechanism that can be taken advantage of for low frequency magnetic fields is the ability of a high permeability material such as mumetal to divert the field by presenting a very low reluctance path to the magnetic flux. Above a few kHz, however, the permeability of such materials is the same as steel.

In actual fact the selection of a shielding material turns out to be less important than the presence of seams, joints and holes in the physical structure of the enclosure. The shielding mechanisms are related to the induction of currents in the shield material, but the currents must be allowed to flow freely. If they have to detour around slots and holes, as shown in Figure 12, the shield loses much of its effectiveness.

As can be seen in Figure 12, the severity of the detour has less to do with the area of the hole than it does with the geometry of the hole. Comparing Figure 12c with 12d shows that a long narrow discontinuity such as a seam can cause more RF leakage than a line of holes with larger total area. A person who is responsible for designing or selecting rack or chassis enclosures for an EMI environment needs to be familiar with the techniques that are available for maintaining electrical continuity across seams. Information on these techniques is available in the references.

Grounds

There are two kinds of grounds: earth-ground and signal ground. The earth is not an equipotential surface, so earth ground potential varies. That and its other electrical properties are not conducive to its use as a return conductor in a circuit. However, circuits are often connected to earth ground for protection against shock hazards. The other kind of ground, signal ground, is an arbitrarily selected reference node in a circuit—the node with respect to which other node voltages in the circuit are measured.

SAFETY GROUND

The standard 3-wire single-phase AC power distribution system is represented in Figure 13. The white wire is earth-grounded at the service entrance. If a load circuit has a metal enclosure or chassis, and if the black wire develops a short to the enclosure, there will be a shock hazard to operating personnel, unless the enclosure itself is earth-grounded. If the enclosure is earth-

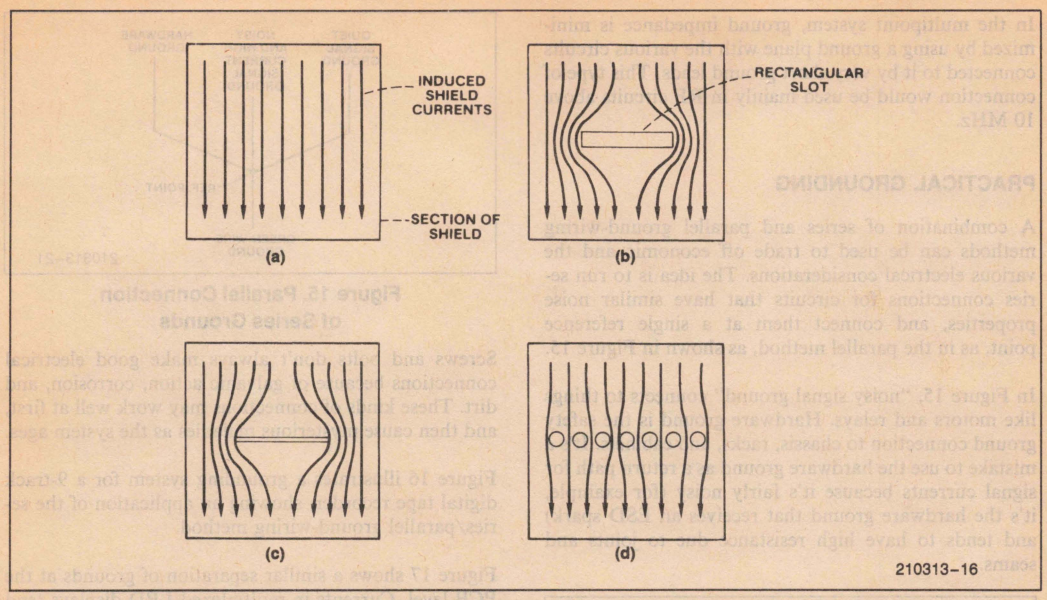


Figure 12. Effect of Shield Discontinuity on Magnetically Induced Shield Current

grounded, a short results in a blown fuse rather than a "hot" enclosure. The earth-ground connection to the enclosure is called a safety ground. The advantage of the 3-wire power system is that it distributes a safety ground along with the power.

Note that the safety-ground wire carries no current, except in case of a fault, so that at least for low frequencies it's at earth-ground potential along its entire length. The white wire, on the other hand, may be several volts off ground, due to the IR drop along its length.

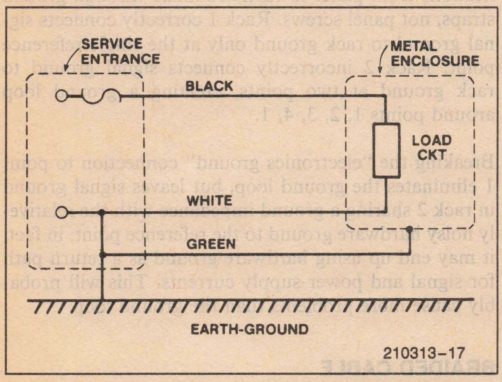


Figure 13. Single-Phase Power Distribution

SIGNAL GROUND

Signal ground is a single point in a circuit that is designated to be the reference node for the circuit. Commonly, wires that connect to this single point are also referred to as "signal ground." In some circles "power supply common" or PSC is the preferred terminology for these conductors. In any case, the manner in which these wires connect to the actual reference point is the basis of distinction among three kinds of signal-ground wiring methods: series, parallel, and multipoint. These methods are shown in Figure 14.

The series connection is pretty common because it's simple and economical. It's the noisiest of the three, however, due to common ground impedance coupling between the circuits. When several circuits share a ground wire, currents from one circuit, flowing through the finite impedance of the common ground line, cause variations in the ground potential of the other circuits. Given that the currents in a digital system tend to be spiked, and that the common impedance is mainly inductive reactance, the variations could be bad enough to cause bit errors in high current or particularly noisy situations.

The parallel connection eliminates common ground impedance problems, but uses a lot of wire. Other disadvantages are that the impedance of the individual ground lines can be very high, and the ground lines themselves can become sources of EMI.

In the multipoint system, ground impedance is minimized by using a ground plane with the various circuits connected to it by very short ground leads. This type of connection would be used mainly in RF circuits above 10 MHz.

PRACTICAL GROUNDING

A combination of series and parallel ground-wiring methods can be used to trade off economic and the various electrical considerations. The idea is to run series connections for circuits that have similar noise properties, and connect them at a single reference point, as in the parallel method, as shown in Figure 15.

In Figure 15, "noisy signal ground" connects to things like motors and relays. Hardware ground is the safety ground connection to chassis, racks, and cabinets. It's a mistake to use the hardware ground as a return path for signal currents because it's fairly noisy (for example, it's the hardware ground that receives an ESD spark) and tends to have high resistance due to joints and seams.

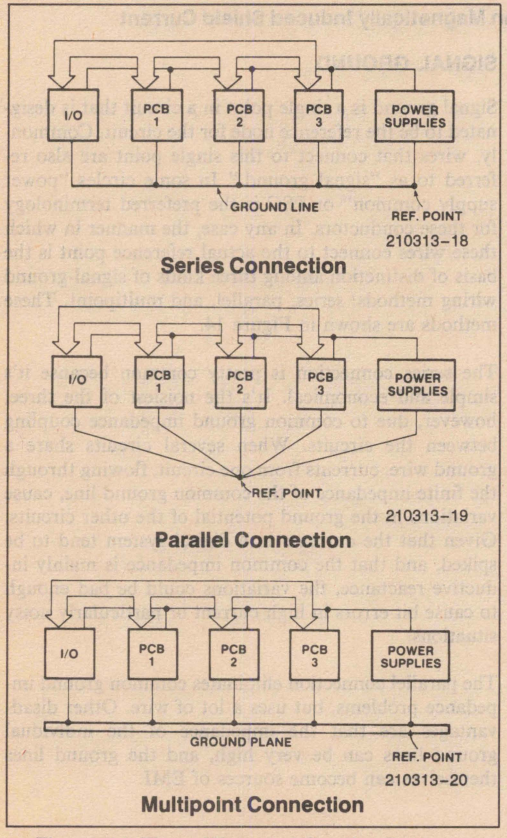


Figure 14. Three Ways to Wire the Grounds

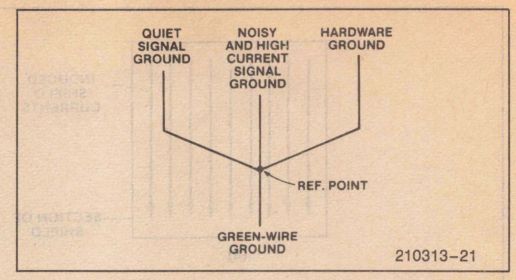


Figure 15. Parallel Connection of Series Grounds

Screws and bolts don't always make good electrical connections because of galvanic action, corrosion, and dirt. These kinds of connections may work well at first, and then cause mysterious maladies as the system ages.

Figure 16 illustrates a grounding system for a 9-track digital tape recorder, showing an application of the series/parallel ground-wiring method.

Figure 17 shows a similar separation of grounds at the PCB level. Currents in multiplexed LED displays tend to put a lot of noise on the ground and supply lines because of the constant switching and changing involved in the scanning process. The segment driver ground is relatively quiet, since it doesn't conduct the LED currents. The digit driver ground is noisier, and should be provided with a separate path to the PCB ground terminal, even if the PCB ground layout is gridded. The LED feed and return current paths should be laid out on opposite sides of the board like parallel flat conductors.

Figure 18 shows right and wrong ways to make ground connections in racks. Note that the safety ground connections from panel to rack are made through ground straps, not panel screws. Rack 1 correctly connects signal ground to rack ground only at the single reference point. Rack 2 incorrectly connects signal ground to rack ground at two points, creating a ground loop around points 1, 2, 3, 4, 1.

Breaking the "electronics ground" connection to point 1 eliminates the ground loop, but leaves signal ground in rack 2 sharing a ground impedance with the relatively noisy hardware ground to the reference point; in fact, it may end up using hardware ground as a return path for signal and power supply currents. This will probably cause more problems than the ground loop.

BRAIDED CABLE

Ground impedance problems can be virtually eliminated by using braided cable. The reduction in impedance is due to skin effect: At higher frequencies the current tends to flow along the surface of a conductor rather

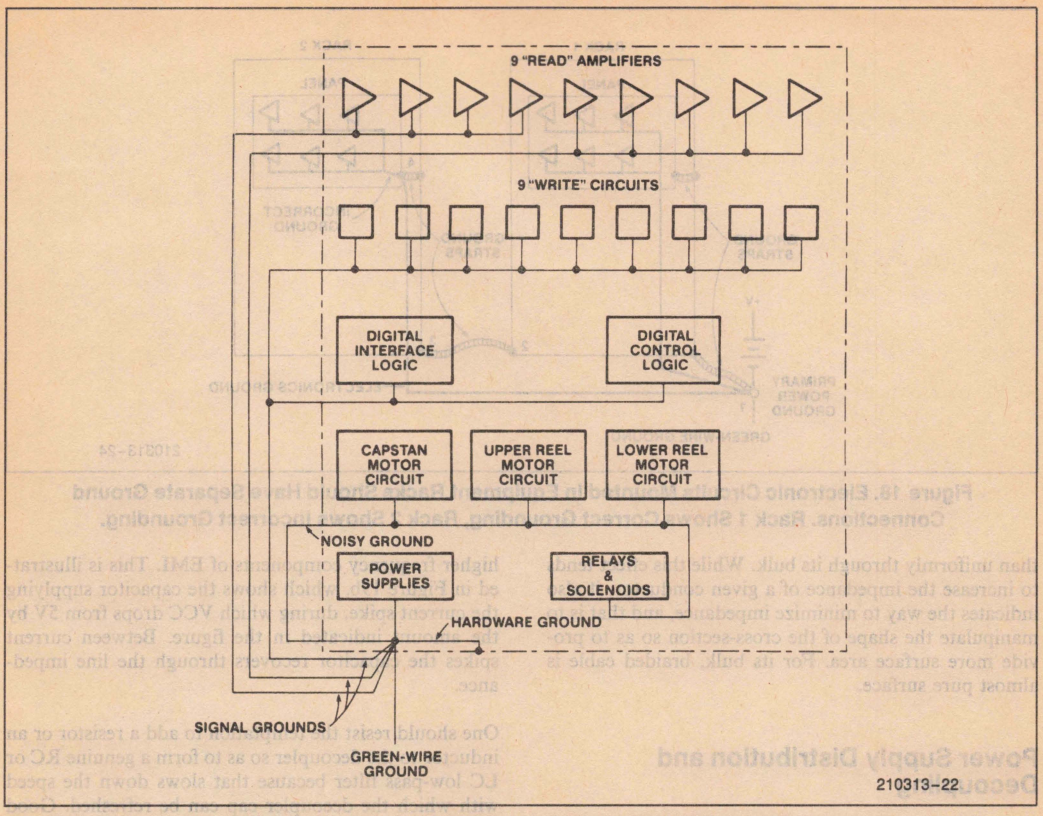


Figure 16. Ground System in a 9-Track Digital Recorder

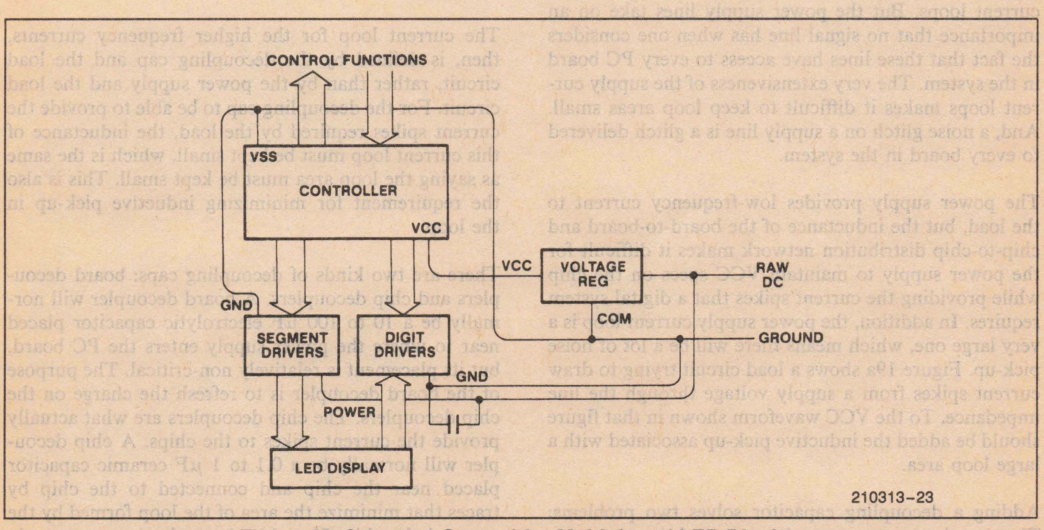


Figure 17. Separate Ground for Multiplexed LED Display

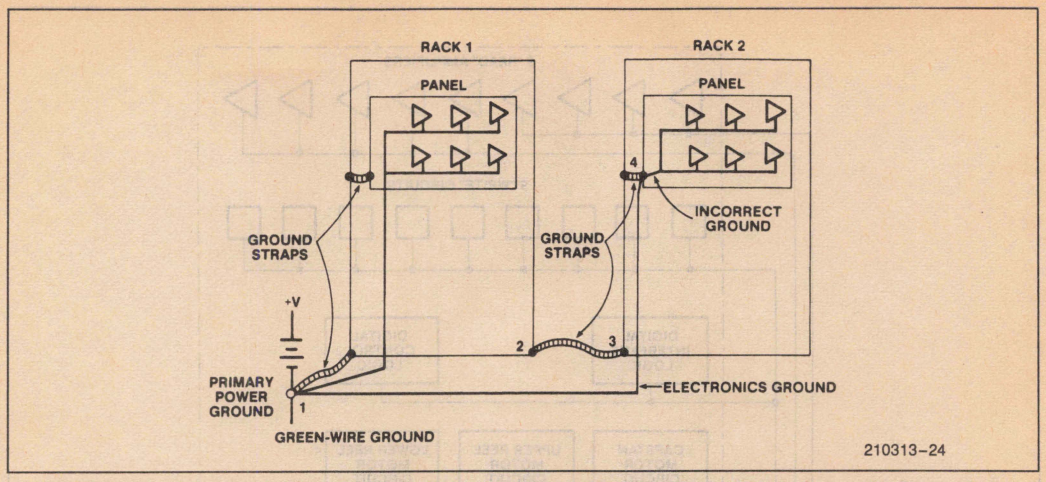


Figure 18. Electronic Circuits Mounted in Equipment Racks Should Have Separate Ground Connections. Rack 1 Shows Correct Grounding, Rack 2 Shows Incorrect Grounding.

than uniformly through its bulk. While this effect tends to increase the impedance of a given conductor, it also indicates the way to minimize impedance, and that is to manipulate the shape of the cross-section so as to provide more surface area. For its bulk, braided cable is almost pure surface.

Power Supply Distribution and Decoupling

The main consideration for power supply distribution lines is, as for signal lines, to minimize the areas of the current loops. But the power supply lines take on an importance that no signal line has when one considers the fact that these lines have access to every PC board in the system. The very extensiveness of the supply current loops makes it difficult to keep loop areas small. And, a noise glitch on a supply line is a glitch delivered to every board in the system.

The power supply provides low-frequency current to the load, but the inductance of the board-to-board and chip-to-chip distribution network makes it difficult for the power supply to maintain VCC specs on the chip while providing the current spikes that a digital system requires. In addition, the power supply current loop is a very large one, which means there will be a lot of noise pick-up. Figure 19a shows a load circuit trying to draw current spikes from a supply voltage through the line impedance. To the VCC waveform shown in that figure should be added the inductive pick-up associated with a large loop area.

Adding a decoupling capacitor solves two problems: The capacitor acts as a nearby source of charge to supply the current spikes through a smaller line impedance, and it defines a much smaller loop area for the higher frequency components of EMI. This is illustrated in Figure 19b, which shows the capacitor supplying the current spike, during which VCC drops from 5V by the amount indicated in the figure. Between current spikes the capacitor recovers through the line impedance.

One should resist the temptation to add a resistor or an inductor to the decoupler so as to form a genuine RC or LC low-pass filter because that slows down the speed with which the decoupler cap can be refreshed. Good filtering and good decoupling are not necessarily the same thing.

The current loop for the higher frequency currents, then, is defined by the decoupling cap and the load circuit, rather than by the power supply and the load circuit. For the decoupling cap to be able to provide the current spikes required by the load, the inductance of this current loop must be kept small, which is the same as saying the loop area must be kept small. This is also the requirement for minimizing inductive pick-up in the loop.

There are two kinds of decoupling caps: board decouplers and chip decouplers. A board decoupler will normally be a 10 to 100 μF electrolytic capacitor placed near to where the power supply enters the PC board, but its placement is relatively non-critical. The purpose of the board decoupler is to refresh the charge on the chip decouplers. The chip decouplers are what actually provide the current spikes to the chips. A chip decoupler will normally be a 0.1 to 1 μF ceramic capacitor placed near the chip and connected to the chip by traces that minimize the area of the loop formed by the cap and the chip. If a chip decoupler is not properly placed on the board, it will be ineffective as a decoupler

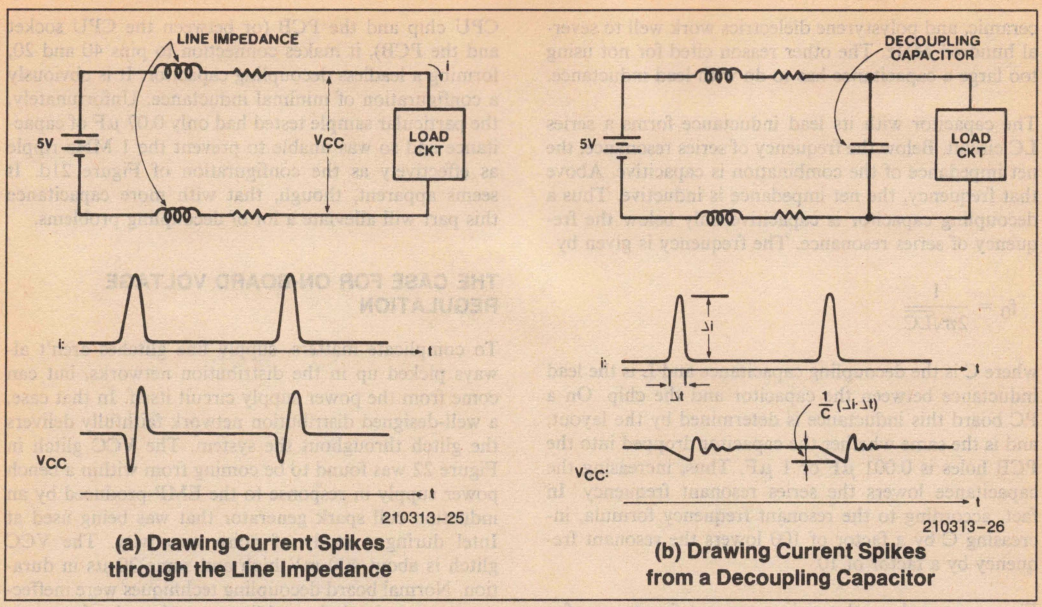


Figure 19. What a Decoupling Capacitor Does

and will serve only to increase the cost of the board. Good and bad placement of decoupling capacitors are illustrated in Figure 20.

Power distribution traces on the PC board need to be laid out so as to obtain minimal area (minimal inductance) in the loops formed by each chip and its decoupler, and by the chip decouplers and the board decoupler. One way to accomplish this goal is to use a power plane. A power plane is the same as a ground plane, but at VCC potential. More economically, a power grid similar to the ground grid previously discussed (Figure 8) can be used. Actually, if the chip decoupling loops are small, other aspects of the power layout are less critical. In other words, power planes and power gridding aren't needed, but power traces *should* be laid in the closest possible proximity to ground traces, preferably so that each power trace is on the direct opposite side of the board from a ground trace.

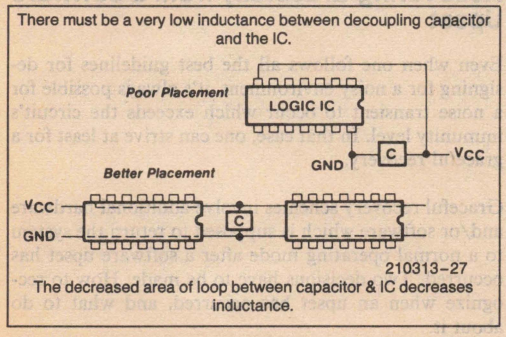


Figure 20. Placement of Decoupling Capacitors

Special-purpose power supply distribution buses which mount on the PCB are available. The buses use a parallel flat conductor configuration, one conductor being a VCC line and the other a ground line. Used in conjunction with a gridded ground layout, they not only provide a low-inductance distribution system, but can themselves form part of the ground grid, thus facilitating the PCB layout. The buses are available with and without enhanced bus capacitance, under the names Mini/Bus® and Q/PAC® from Rogers Corp. (5750 E. McKellips, Mesa, AZ 85205).

SELECTING THE VALUE OF THE DECOUPLING CAP

The effectiveness of the decoupling cap has a lot to do with the way the power and ground traces connect this capacitor to the chip. In fact, the area formed by this loop is more important than the value of the capacitance. Then, given that the area of this loop is indeed minimal, it can generally be said that the larger the value of the decoupling cap, the more effective it is, if the cap has a mica, ceramic, glass, or polystyrene dielectric.

It's often said, and not altogether accurately, that the chip decoupler shouldn't have too large a value. There are two reasons for this statement. One is that some capacitors, because of the nature of their dielectrics, tend to become inductive or lossy at higher frequencies. This is true of electrolytic capacitors, but mica, glass,

ceramic, and polystyrene dielectrics work well to several hundred MHz. The other reason cited for not using too large a capacitance has to do with lead inductance.

The capacitor with its lead inductance forms a series LC circuit. Below the frequency of series resonance, the net impedance of the combination is capacitive. Above that frequency, the net impedance is inductive. Thus a decoupling capacitor is capacitive only below the frequency of series resonance. The frequency is given by

$$f_0 = \frac{1}{2\pi\sqrt{LC}}$$

where C is the decoupling capacitance and L is the lead inductance between the capacitor and the chip. On a PC board this inductance is determined by the layout, and is the same whether the capacitor dropped into the PCB holes is 0.001 μF or 1 μF. Thus, increasing the capacitance lowers the series resonant frequency. In fact, according to the resonant frequency formula, increasing C by a factor of 100 lowers the resonant frequency by a factor of 10.

Figures quoted on the series resonant frequency of a 0.01 μF capacitor run from 10 to 15 MHz, depending on the lead length. If these numbers were accurate, a 1 μF capacitor in the same position on the board would have a resonant frequency of 1.0 to 1.5 MHz, and as a decoupler would do more harm than good. However, the numbers are based on a presumed inductance of a given length of wire (the lead length). It should be noted that a "length of wire" has no inductance at all, strictly speaking. Only a complete current loop has inductance, and the inductance depends on the geometry of the loop. Figures quoted on the inductance of a length of wire are based on a presumably "very large" loop area, such that the magnetic field produced by the return current has no cancellation effect on the field produced by the current in the given length of wire. Such a loop geometry is not and should not be the case with the decoupling loop.

Figure 21 shows VCC waveforms, measured between pins 40 and 20 (VCC and VSS) of an 8751 CPU, for several conditions of decoupling on a PC board that has a decoupling loop area slightly larger than necessary. These photographs show the effects of increasing the decoupling capacitance and decreasing the area of the decoupling loop. The indications are that a 1 μF capacitor is better than a 0.1 μF capacitor, which in turn is better than nothing, and that the board should have been laid out with more attention paid to the area of the decoupling loop.

Figure 21e was obtained using a special-purpose experimental capacitor designed by Rogers Corp. (Q-Pac Division, Mesa, AZ) for use as a decoupler. It consists of two parallel plates, the length of a 40-pin DIP, separated by a ceramic dielectric. Sandwiched between the CPU chip and the PCB (or between the CPU socket and the PCB), it makes connection to pins 40 and 20, forming a leadless decoupling capacitor. It is obviously a configuration of minimal inductance. Unfortunately, the particular sample tested had only 0.07 μF of capacitance and so was unable to prevent the 1 MHz ripple as effectively as the configuration of Figure 21d. It seems apparent, though, that with more capacitance this part will alleviate a lot of decoupling problems.

THE CASE FOR ON-BOARD VOLTAGE REGULATION

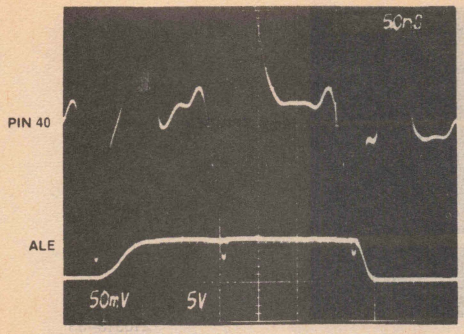
To complicate matters, supply line glitches aren't always picked up in the distribution networks, but can come from the power supply circuit itself. In that case, a well-designed distribution network faithfully delivers the glitch throughout the system. The VCC glitch in Figure 22 was found to be coming from within a bench power supply in response to the EMP produced by an induction coil spark generator that was being used at Intel during a study of noise sensitivity. The VCC glitch is about 400 mV high and some 20 μs in duration. Normal board decoupling techniques were ineffective in removing it, but adding an on-board voltage regulator chip did the job.

Thus, a good case can be made in favor of using a voltage regulator chip on each PCB, instead of doing all the voltage regulation at the supply circuit. This eases requirements on the heat-sinking at the supply circuit, and alleviates much of the distribution and board decoupling headaches. However, it also brings in the possibility that different boards would be operating at slightly different VCC levels due to tolerance in the regulator chips; this then leads to slightly different logic levels from board to board. The implications of that may vary from nothing to latch-up, depending on what kinds of chips are on the boards, and how they react to an input "high" that is perhaps 0.4V higher than local VCC.

Recovering Gracefully from a Software Upset

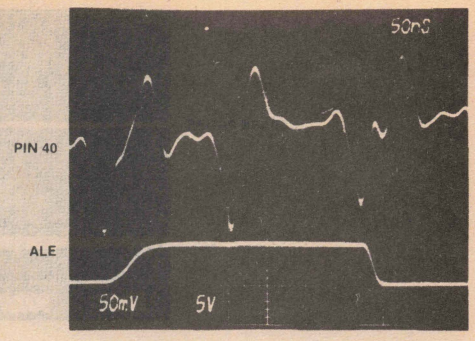
Even when one follows all the best guidelines for designing for a noisy environment, it's always possible for a noise transient to occur which exceeds the circuit's immunity level. In that case, one can strive at least for a graceful recovery.

Graceful recovery schemes involve additional hardware and/or software which is supposed to return the system to a normal operating mode after a software upset has occurred. Two decisions have to be made: How to recognize when an upset has occurred, and what to do about it.



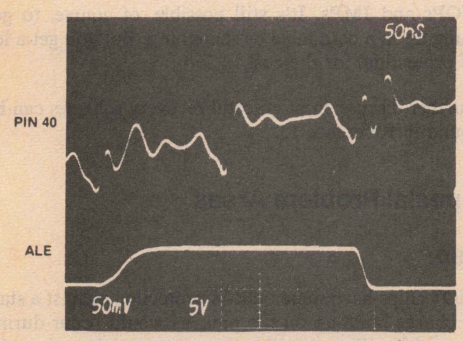
210313-28

(a) No Decoupling Cap



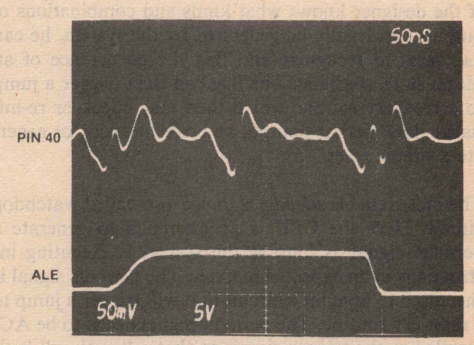
210313-29

(b) 0.1 μF Decoupler in Place on the PCB



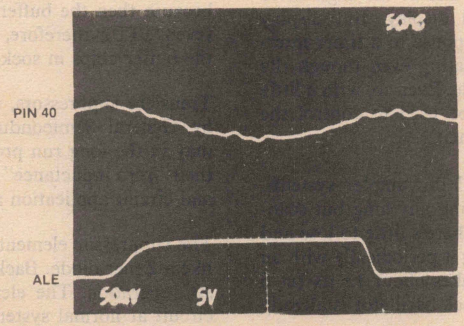
210313-30

(c) 0.1 μF Decoupler Stretched Directly from Pin 40 to Pin 20, under the Socket. (The difference between this and 21b is due only to the change in loop geometry. Also shown is the upward slope of a ripple in VCC. The ripple frequency is 1 MHz, the same as ALE.)



210313-31

(d) 1.0 μF Decoupler Stretched Directly from Pin 40 to Pin 20, under the Socket. (This prevents the 1 MHz ripple, but there's no reduction in higher frequency components. Further increases in capacitance effected no further improvement.)



210313-32

(e) Special-Purpose Decoupling Cap under Development by Rogers Corp. (Further discussion in text.)

Figure 21. Noise on VCC Line

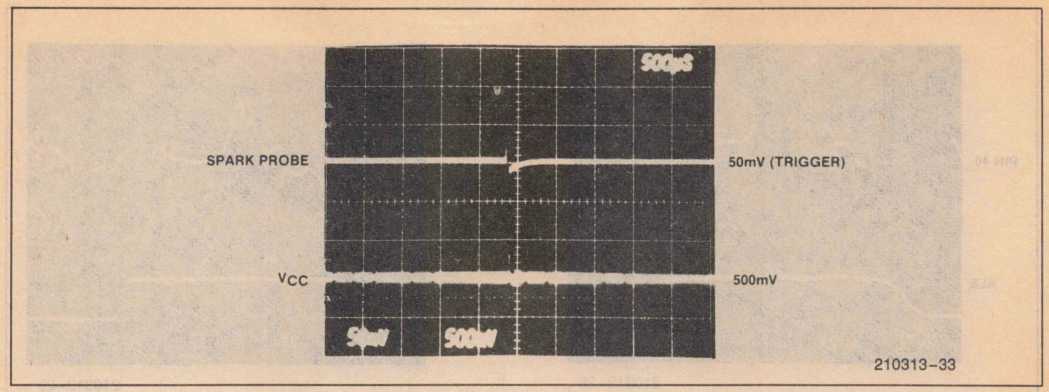


Figure 22. EMP-Induced Glitch

If the designer knows what kinds and combinations of outputs can legally be generated by the system, he can use gates to recognize and flag the occurrence of an illegal state of affairs. The flag can then trigger a jump to a recovery routine which then may check or re-initialize data, perhaps output an error message, or generate a simple reset.

The most reliable scheme is to use a so-called watchdog circuit. Here the CPU is programmed to generate a periodic signal as long as the system is executing instructions in an expected manner. The periodic signal is then used to hold off a circuit that will trigger a jump to a recovery routine. The periodic signal needs to be AC-coupled to the trigger circuit so that a "stuck-at" fault won't continue to hold off the trigger. Then, if the processor locks up someplace, the periodic signal is lost and the watchdog triggers a reset.

In practice, it may be convenient to drive the watchdog circuit with a signal which is being generated anyway by the system. One needs to be careful, however, that an upset does in fact discontinue that signal. Specifically, for example, one could use one of the digit drive signals going to a multiplexed display. But display scanning is often handled in response to a timer-interrupt, which may continue operating even though the main program is in a failure mode. Even so, with a little extra software, the signal can be used to control the watchdog (see Reference 8 on this).

Simpler schemes can work well for simpler systems. For example, if a CPU isn't doing anything but scanning and decoding a keyboard, there's little to lose and much to gain by simply resetting it periodically with an astable multivibrator. It only takes about 13 μs (at 6 MHz) to reset an 8048 if the clock oscillator is already running.

A zero-cost measure is simply to fill all unused program memory with NOPs and JMPs to a recovery routine. The effectiveness of this method is increased by writing the program in segments that are separated by NOPs and JMPs. It's still possible, of course, to get hung up in a data table or something. But you get a lot of protection, for the cost.

Further discussion of graceful recovery schemes can be found in Reference 13.

Special Problem Areas

ESD

MOS chips have some built-in protection against a static charge build-up on the pins, as would occur during normal handling, but there's no protection against the kinds of current levels and rise times that occur in a genuine electrostatic spark. These kinds of discharges can blow a crater in the silicon.

It must be recognized that connecting CPU pins unprotected to a keyboard or to anything else that is subject to electrostatic discharges makes an extremely fragile configuration. Buffering them is the very least one can do. But buffering doesn't completely solve the problem, because then the buffer chips will sustain the damage (even TTL); therefore, one might consider mounting the buffer chips in sockets for ease of replacement.

Transient suppressors, such as the TranZorbs® made by General Semiconductor Industries (Tempe, AZ), may in the long run provide the cheapest protection if their "zero inductance" structure is used. The structure and circuit application are shown in Figure 23.

The suppressor element is a pn junction that operates like a Zener diode. Back-to-back units are available for AC operation. The element is more or less an open circuit at normal system voltage (the standoff voltage rating for the device), and conducts like a Zener diode at the clamping voltage.

The lead inductance in the conventional transient suppressor package makes the conventional package essen-

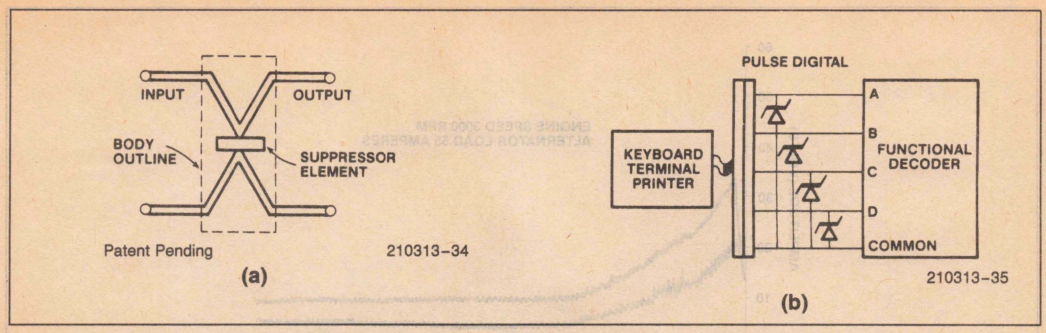


Figure 23. "Zero-Inductance" Structure and Use in Circuit

tially useless for protection against ESD pulses, owing to the fast rise of these pulses. The "zero inductance" units are available singly in a 4-pin DIP, and in arrays of four to a 16-pin DIP for PCB level protection. In that application they should be mounted in close proximity to the chips they protect.

In addition, metal enclosures or frames or parts that can receive an ESD spark should be connected by braided cable to the green-wire ground. Because of the ground impedance, ESD current shouldn't be allowed to flow through any signal ground, even if the chips are protected by transient suppressors. A 35 kV ESD spark can always spare a few hundred volts to drive a fast current pulse down a signal ground line if it can't find a braided cable to follow. Think how delighted your 8048 will be to find its VSS pin 250V higher than VCC for a few 10s of nanoseconds.

THE AUTOMOTIVE ENVIRONMENT

The automobile presents an extremely hostile environment for electronic systems. There are several parts to it:

1. Temperature extremes from −40°C to +125°C (under the hood) or +85°C (in the passenger compartment)
2. Electromagnetic pulses from the ignition system
3. Supply line transients that will knock your socks off

One needs to take a long, careful look at the temperature extremes. The allowable storage temperature range for most Intel MOS chips is −65°C to +150°C, although some chips have a maximum storage temperature rating of +125°C. In operation (or "under bias," as the data sheets say) the allowable ambient temperature range depends on the product grade, as follows:

Grade	Ambient Temperature	
	Min	Max
Commercial	0	70
Industrial	−40	+85
Automotive	−40	+110
Military	−55	+125

The different product grades are actually the same chip, but tested according to different standards. Thus, a given commercial-grade chip might actually pass military temperature requirements, but not have been tested for it. (Of course, there are other differences in grading requirements having to do with packaging, burn-in, traceability, etc.)

In any case, it's apparent that commercial-grade chips can't be used safely in automotive applications, not even in the passenger compartment. Industrial-grade chips can be used in the passenger compartment, and automotive or military chips are required in under-the-hood applications.

Ignition noise, CB radios, and that sort of thing are probably the least of your worries. In a poorly designed system, or in one that has not been adequately tested for the automotive environment, this type of EMI might cause a few software upsets, but not destroy chips.

The major problem, and the one that seems to come as the biggest surprise to most people, is the line transients. Regrettably, the 12V battery is not actually the source of power when the car is running. The charging system is, and it's not very clean. The only time the battery is the real source of power is when the car is first being started, and in that condition the battery terminals may be delivering about 5V or 6V. As follows is a brief description of the major idiosyncracies of the "12V" automotive power line.

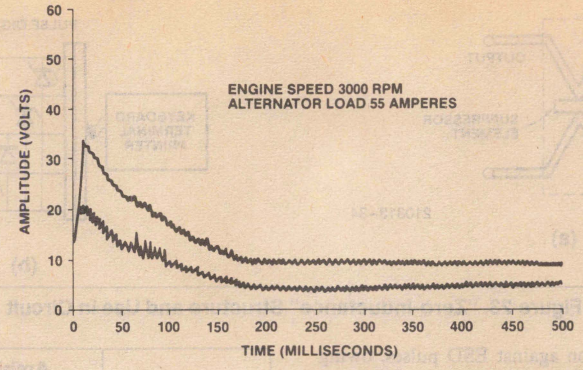


Figure 24. Typical Load Dump Transients

- An abrupt reduction in the alternator load causes a positive voltage transient called "load dump." In a load dump transient the line voltage rises to 20V or 30V in a few μs, then decays exponentially with a time constant of about 100 μs, as shown in Figure 24. Much higher peak voltages and longer decay times have also been reported. The worst case load dump is caused by disconnecting a low battery from the alternator circuit while the alternator is running. Normally this would happen intermittently when the battery terminal connections are defective.
- When the ignition is turned off, as the field excitation decays, the line voltage can go to between -40V and -100V for 100 μs or more.
- Miscellaneous solenoid switching transients, such as the one shown in Figure 25, can drive the line to $+$ or -200V to 400V for several μs.
- Mutual coupling between unshielded wires in long harnesses can induce 100V and 200V transients in unprotected circuits.

What all this adds up to is that people in the business of building systems for automotive applications need a comprehensive testing program. An SAE guideline which describes the automotive environment is available to designers: SAE J1211, "Recommended Environmental Practices for Electronic Equipment Design," *1980 SAE Handbook,* Part 1, pp. 22.80–22.96.

Some suggestions for protecting circuitry are shown in Figure 26. A transient suppressor is placed in front of the regulator chip to protect it. Since the rise times in these transients are not like those in ESD pulses, lead inductance is less critical and conventional devices can be used. The regulator itself is pretty much of a necessity, since a load dump transient is simply not going to be removed by any conventional LC or RC filter.

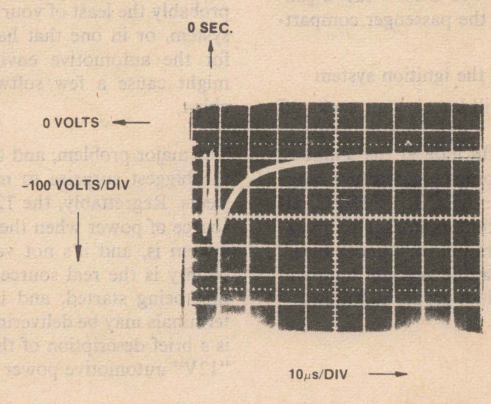


Figure 25. Transient Created by De-energizing an Air Conditioning Clutch Solenoid

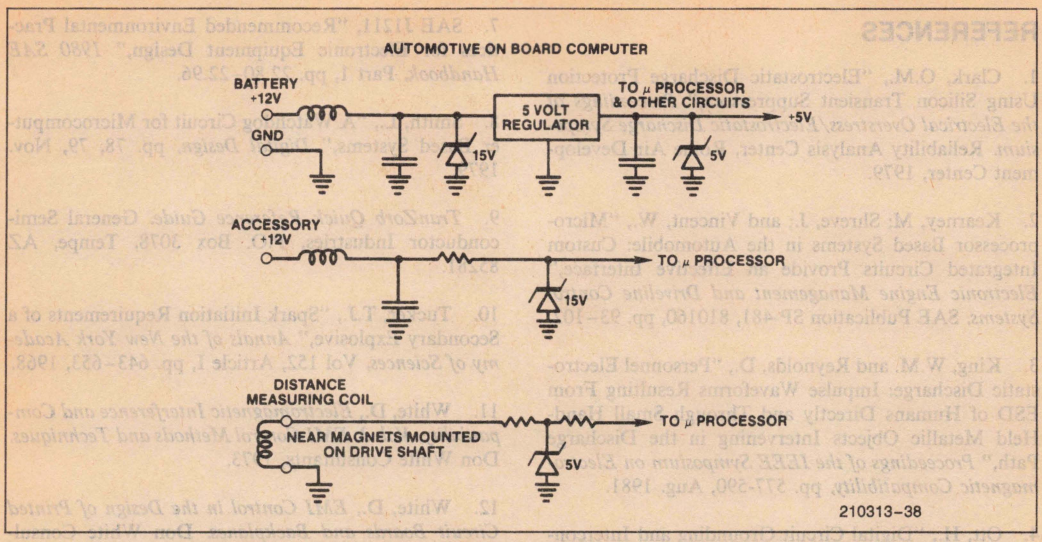


Figure 26. Use of Transient Suppressors in Automotive Applications

Special I/O interfacing is also required, because of the need for high tolerance to voltage transients, input noise, input/output isolation, etc. In addition, switches that are being monitored or driven by these buffers are usually referenced to chassis ground instead of signal ground, and in a car there can be many volts difference between the two. I/O interfacing is discussed in Reference 2.

Parting Thoughts

The main sources of information for this Application Note were the references by Ott and by White. Reference 5 is probably the finest treatment currently available on the subject. The other references provided specific information as cited in the text.

Courses and seminars on the subject of electromagnetic interference are given regularly throughout the year. Information on these can be obtained from:

IEEE Electromagnetic Compatibility Society
EMC Education Committee
345 East 47th Street
New York, NY 10017

Don White Consultants, Inc.
International Training Centre
P.O. Box D
Gainesville, VA 22065
Phone: (703) 347-0030

The EMC Education committee has available a video tape: "Introduction to EMC—A Video Training Tape," by Henry Ott. Don White Consultants offers a series of training courses on many different aspects of electromagnetic compatibility. Most organizations that sponsor EMC courses also offer in-plant presentations.

REFERENCES

1. Clark, O.M., "Electrostatic Discharge Protection Using Silicon Transient Suppressors," *Proceedings of the Electrical Overstress/Electrostatic Discharge Symposium.* Reliability Analysis Center, Rome Air Development Center, 1979.

2. Kearney, M; Shreve, J.; and Vincent, W., "Microprocessor Based Systems in the Automobile: Custom Integrated Circuits Provide an Effective Interface," *Electronic Engine Management and Driveline Control Systems,* SAE Publication SP-481, 810160, pp. 93–102.

3. King, W.M. and Reynolds, D., "Personnel Electrostatic Discharge: Impulse Waveforms Resulting From ESD of Humans Directly and Through Small Hand-Held Metallic Objects Intervening in the Discharge Path," *Proceedings of the IEEE Symposium on Electromagnetic Compatibility,* pp. 577-590, Aug. 1981.

4. Ott, H., "Digital Circuit Grounding and Interconnection," *Proceedings of the IEEE Symposium on Electromagnetic Compatibility,* pp. 292-297, Aug. 1981.

5. Ott, H., *Noise Reduction Techniques in Electronic Systems.* New York: Wiley, 1976.

6. *1981 Interference Technology Engineers' Master (ITEM) Directory and Design Guide.* R. and B. Enterprises, P.O. Box 328, Plymouth Meeting, PA 19426.

7. SAE J1211, "Recommended Environmental Practices for Electronic Equipment Design," *1980 SAE Handbook,* Part 1, pp. 22.80–22.96.

8. Smith, L., "A Watchdog Circuit for Microcomputer Based Systems," *Digital Design,* pp. 78, 79, Nov. 1979.

9. *TranZorb Quick Reference Guide.* General Semiconductor Industries, P.O. Box 3078, Tempe, AZ 85281.

10. Tucker, T.J., "Spark Initiation Requirements of a Secondary Explosive," *Annals of the New York Academy of Sciences,* Vol 152, Article I, pp. 643–653, 1968.

11. White, D., *Electromagnetic Interference and Compatibility, Vol. 3: EMI Control Methods and Techniques.* Don White Consultants, 1973.

12. White, D., *EMI Control in the Design of Printed Circuit Boards and Backplanes.* Don White Consultants, 1981.

13. Yarkoni, B. and Wharton, J., "Designing Reliable Software for Automotive Applications," *SAE Transactions,* 790237, July 1979.

APPLICATION NOTE

AP-155

August 1986

Oscillators for Microcontrollers

TOM WILLIAMSON
MICROCONTROLLER
TECHNICAL MARKETING

Order Number: 230659-001

INTRODUCTION

Intel's microcontroller families (MCS®-48, MCS®-51, and iACX-96) contain a circuit that is commonly referred to as the "on-chip oscillator". The on-chip circuitry is not itself an oscillator, of course, but an amplifier that is suitable for use as the amplifier part of a feedback oscillator. The data sheets and Microcontoller Handbook show how the on-chip amplifier and several off-chip components can be used to design a working oscillator. With proper selection of off-chip components, these oscillator circuits will perform better than almost any other type of clock oscillator, and by almost any criterion of excellence. The suggested circuits are simple, economical, stable, and reliable.

We offer assistance to our customers in selecting suitable off-chip components to work with the on-chip oscillator circuitry. It should be noted, however, that Intel cannot assume the responsibility of writing specifications for the off-chip components of the complete oscillator circuit, nor of guaranteeing the performance of the finished design in production, anymore than a transistor manufacturer, whose data sheets show a number of suggested amplifier circuits, can assume responsibility for the operation, in production, of any of them.

We are often asked why we don't publish a list of required crystal or ceramic resonator specifications, and recommend values for the other off-chip components. This has been done in the past, but sometimes with consequences that were not intended.

Suppose we suggest a maximum crystal resistance of 30 ohms for some given frequency. Then your crystal supplier tells you the 30-ohm crystals are going to cost twice as much as 50-ohm crystals. Fearing that Intel will not "guarantee operation" with 50-ohm crsytals, you order the expensive ones. In fact, Intel guarantees only what is embodied within an Intel product. Besides, there is no reason why 50-ohm crystals couldn't be used, if the other off-chip components are suitably adjusted.

Should we recommend values for the other off-chip components? Should we do it for 50-ohm crystals or 30-ohm crystals? With respect to what should we optimize their selection? Should we minimize start-up time or maximize frequency stability? In many applications, neither start-up time nor frequency stability are particularly critical, and our "recommendations" are only restricting your system to unnecessary tolerances. It all depends on the application.

Although we will neither "specify" nor "recommend" specific off-chip components, we do offer assistance in these tasks. Intel application engineers are available to provide whatever technical assistance may be needed or desired by our customers in designing with Intel products.

This Application Note is intended to provide such assistance in the design of oscillator circuits for microcontroller systems. Its purpose is to describe in a practical manner how oscillators work, how crystals and ceramic resonators work (and thus how to spec them), and what the on-chip amplifier looks like electronically and what its operating characteristics are. A BASIC program is provided in Appendix II to assist the designer in determining the effects of changing individual parameters. Suggestions are provided for establishing a pre-production test program.

FEEDBACK OSCILLATORS

Loop Gain

Figure 1 shows an amplifier whose output line goes into some passive network. If the input signal to the amplifier is v_1, then the output signal from the amplifer is $v_2 = Av_1$ and the output signal from the passive network is $v_3 = \beta v_2 = \beta A v_1$. Thus βA is the overall gain from terminal 1 to terminal 3.

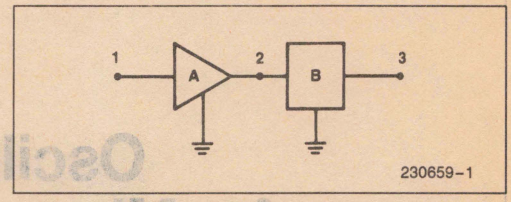


Figure 1. Factors in Loop Gain

Now connect terminal 1 to terminal 3, so that the signal path forms a loop: 1 to 2 to 3, which is also 1. Now we have a feedback loop, and the gain factor βA is called the *loop gain.*

Gain factors are complex numbers. That means they have a magnitude and a phase angle, both of which vary with frequency. When writing a complex number, one must specify both quantities, magnitude and angle. A number whose magnitude is 3, and whose angle is 45 degrees is commonly written this way: $3\angle 45°$. The number 1 is, in complex number notation, $1\angle 0°$, while -1 is $1\angle 180°$.

By closing the feedback loop in Figure 1, we force the equality

$$v_1 = \beta A v_1$$

This equation has two solutions:

1) $v_1 = 0$;

2) $\beta A = 1\angle 0°$.

In a given circuit, either or both of the solutions may be in effect. In the first solution the circuit is quiescent (no output signal). If you're trying to make an oscillator, a no-signal condition is unacceptable. There are ways to guarantee that the second solution is the one that will be in effect, and that the quiescent condition will be excluded.

How Feedback Oscillators Work

A feedback oscillator amplifies its own noise and feeds it back to itself in exactly the right phase, at the oscillation frequency, to build up and reinforce the desired oscillations. Its ability to do that depends on its loop gain. First, oscillations can occur only at the frequency for which the loop gain has a phase angle of 0 degrees. Second build-up of oscillations will occur only if the loop gain exceeds 1 at the frequency. Build-up continues until nonlinearities in the circuit reduce the average value of the loop gain to exactly 1.

Start-up characteristics depend on the small-signal properties of the circuit, specifically, the small-signal loop gain. Steady-state characteristics of the oscillator depend on the large-signal properties of the circuit, such as the transfer curve (output voltage vs. input voltage) of the amplifier, and the clamping effect of the input protection devices. These things will be discussed more fully further on. First we will look at the basic operation of the particular oscillator circuit, called the "positive reactance" oscillator.

The Positive Reactance Oscillator

Figure 2 shows the configuration of the positive reactance oscillator. The inverting amplifier, working into the impedance of the feedback network, produces an output signal that is nominally 180 degrees out of phase with its input. The feedback network must provide an additional 180 degrees phase shift, such that the overall loop gain has zero (or 360) degrees phase shift at the oscillation frequency.

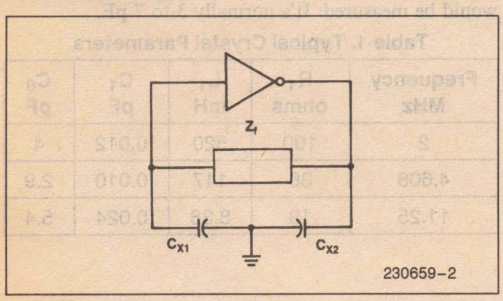


Figure 2. Positive Reactance Oscillator

In order for the loop gain to have zero phase angle it is necessary that the feedback element Z_f have a positive reactance. That is, it must be inductive. Then, the frequency at which the phase angle is zero is approximately the frequency at which

$$X_f = \frac{+1}{\omega C}$$

where X_f is the reactance of Z_f (the total Z_f being $R_f + jX_f$, and C is the series combination of C_{X1} and C_{X2}.

$$C = \frac{C_{X1}\,C_{X2}}{C_{X1} + C_{X2}}$$

In other words, Z_f and C form a parallel resonant circuit.

If Z_f is an inductor, then $X_f = \omega L$, and the frequency at which the loop gain has zero phase is the frequency at which

$$\omega L = \frac{1}{\omega C}$$

or

$$\omega = \frac{1}{\sqrt{LC}}$$

Normally, Z_f is not an inductor, but it must still have a positive reactance in order for the circuit to oscillate. There are some piezoelectric devices on the market that show a positive reactance, and provide a more stable oscillation frequency than an inductor will. Quartz crystals can be used where the oscillation frequency is critical, and lower cost ceramic resonators can be used where the frequency is less critical.

When the feedback element is a piezoelectric device, this circuit configuration is called a Pierce oscillator. The advantage of piezoelectric resonators lies in their property of providing a wide range of positive reactance values over a very narrow range of frequencies. The reactance will equal $1/\omega C$ at some frequency within this range, so the oscillation frequency will be within the same range. Typically, the width of this range is

only 0.3% of the nominal frequency of a quartz crystal, and about 3% of the nominal frequency of a ceramic resonator. With relatively little design effort, frequency accuracies of 0.03% or better can be obtained with quartz crystals, and 0.3% or better with ceramic resonators.

QUARTZ CRYSTALS

The crystal resonator is a thin slice of quartz sandwiched between two electrodes. Electrically, the device looks pretty much like a 5 or 6 pF capacitor, except that over certain ranges of frequencies the crystal has a positive (i.e., inductive) reactance.

The ranges of positive reactance originate in the piezoelectric property of quartz: Squeezing the crystal generates an internal E-field. The effect is reversible: Applying an *AC* E-field causes the crystal to vibrate. At certain vibrational frequencies there is a mechanical resonance. As the E-field frequency approaches a frequency of mechanical resonance, the measured reactance of the crystal becomes positive, as shown in Figure 3.

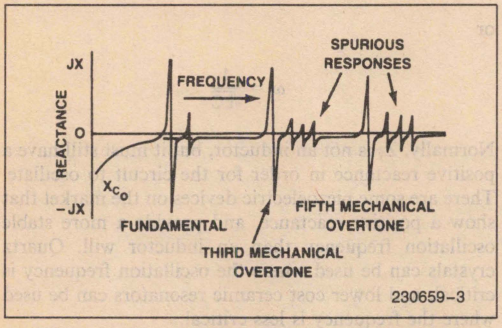


Figure 3. Crystal Reactance vs. Frequency

Typically there are several ranges of frequencies wherein the reactance of the crystal is positive. Each range corresponds to a different mode of vibration in the crystal. The main resonsances are the so-called fundamental response and the third and fifth overtone responses.

The overtone responses shouldn't be confused with the harmonics of the fundamental. They're not harmonics, but different vibrational modes. They're not in general at exact integer multiples of the fundamental frequency. There will also be "spurious" responses, occurring typically a few hundred KHz above each main response.

To assure that an oscillator starts in the desired mode on power-up, something must be done to suppress the loop gain in the undesired frequency ranges. The crystal itself provides some protection against unwanted modes of oscillation; too much resistance in that mode, for example. Additionally, junction capacitances in the amplifying devices tend to reduce the gain at higher frequencies, and thus may discriminate against unwanted modes. In some cases a circuit fix is necessary, such as inserting a trap, a phase shifter, or ferrite beads to kill oscillations in unwanted modes.

Crystal Parameters

Equivalent Circuit

Figure 4 shows an equivalent circuit that is used to represent the crystal for circuit analysis.

The R_1-L_1-C_1 branch is called the motivational arm of the crystal. The values of these parameters derive from the mechanical properties of the crystal and are constant for a given mode of vibration. Typical values for various nominal frequencies are shown in Table 1.

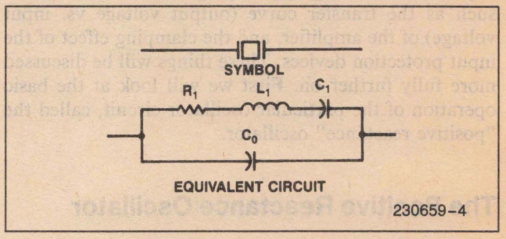


Figure 4. Quartz Crystal: Symbol and Equivalent Circuit

C_0 is called the shunt capacitance of the crystal. This is the capacitance of the crystal's electrodes and the mechanical holder. If one were to measure the reactance of the crystal at a freuqency far removed from a resonance frequency, it is the reactance of this capacitance that would be measured. It's normally 3 to 7 pF.

Table 1. Typical Crystal Parameters

Frequency MHz	R_1 ohms	L_1 mH	C_1 pF	C_0 pF
2	100	520	0.012	4
4.608	36	117	0.010	2.9
11.25	19	8.38	0.024	5.4

The series resonant frequency of the crystal is the frequency at which L_1 and C_1 are in resonance. This frequency is given by

$$f_s = \frac{1}{2\pi\sqrt{L_1 C_1}}$$

At this frequency the impedance of the crystal is R_1 in parallel with the reactance of C_0. For most purposes, this impedance is taken to be just R_1, since the reactance of C_0 is so much larger than R_1.

Load Capacitance

A crystal oscillator circuit such as the one shown in Figure 2 (redrawn in Figure 5) operates at the frequency for which the crystal is antiresonant (ie, parallel-resonant) with the total capacitance across the crystal terminals external to the crystal. This total capacitance external to the crystal is called the load capacitance.

As shown in Figure 5, the load capacitance is given by

$$C_L = \frac{C_{X1}\,C_{X2}}{C_{X1} + C_{X2}} + C_{stray}$$

The crystal manufacturer needs to know the value of C_L in order to adjust the crystal to the specified frequency.

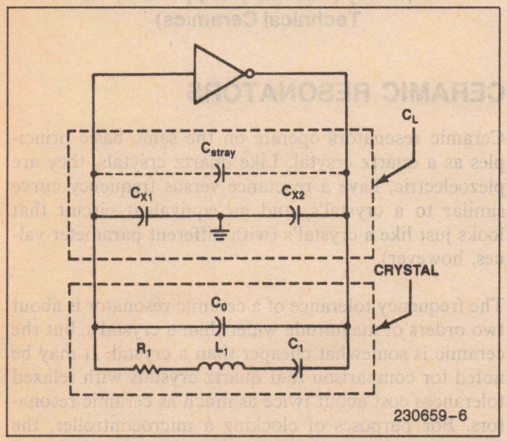


Figure 5. Load Capacitance

The adjustment involves putting the crystal in *series* with the specified C_L, and then "trimming" the crystal to obtain resonance of the series combination of the crystal and C_L at the specified frequency. Because of the high Q of the crystal, the resonant frequency of the series combination of the crystal and C_L is the same as the antiresonant frequency of the *parallel* combination of the crystal and C_L. This frequency is given by

$$f_a = \frac{1}{2\pi\sqrt{L_1 C_1 (C_L + C_0)/(C_1 + C_L + C_0)}}$$

These frequency formulas are derived (in Appendix A) from the equivalent circuit of the crystal, using the assumptions that the Q of the crystal is extremely high, and that the circuit external to the crystal has no effect on the frequency other than to provide the load capacitance C_L. The latter assumption is not precisely true, but it is close enough for present purposes.

"Series" vs. "Parallel" Crystals

There is no such thing as a "series cut" crystal as opposed to a "parallel cut" crystal. There are different cuts of crystal, having to do with the parameters of its motional arm in various frequency ranges, but there is no special cut for series or parallel operation.

An oscillator is series resonant if the oscillation frequency is f_s of the crystal. To operate the crystal at f_s, the amplifier has to be noninverting. When buying a crystal for such an oscillator, one does not specify a load capacitance. Rather, one specifies the loading condition as "series."

If a "series" crystal is put into an oscillator that has an inverting amplifier, it will oscillate in parallel resonance with the load capacitance presented to the crystal by the oscillator circuit, at a frequency slightly above f_s. In fact, at approximately

$$f_a = f_s\left(1 + \frac{C_1}{2(C_L + C_0)}\right)$$

This frequency would typically be about 0.02% above f_s.

Equivalent Series Resistance

The "series resistance" often listed on quartz crystal data sheets is the real part of the crystal impedance at the crystal's calibration frequency. This will be R1 if the calibration frequency is the series resonant frequency of the crystal. If the crystal is calibrated for parallel resonance with a load capacitance CL, the equivalent series resistance will be

$$ESR = R_1\left(1 + \frac{C_0}{C_L}\right)^2$$

The crystal manufacturer measures this resistance at the calibration frequency during the same operation in which the crystal is adjusted to the calibration frequency.

Frequency Tolerance

Frequency tolerance as discussed here is not a requirement on the crystal, but on the complete oscillator. There are two types of frequency tolerances on oscillators: frequency *acccuracy* and frequency *stability*. Frequency accuracy refers to the oscillator's ability to run at an exact specified frequency. Frequency stability refers to the constancy of the oscillation frequency.

Frequency accuracy requires mainly that the oscillator circuit present to the crystal the same load capacitance that it was adjusted for. Frequency stability requires mainly that the load capacitance be constant.

In most digital applications the accuracy and stability requirements on the oscillator are so wide that it makes very little difference what load capacitance the crystal was adjusted to, or what load capacitance the circuit actually presents to the crystal. For example, if a crystal was calibrated to a load capacitance of 25 pF, and is used in a circuit whose actual load capacitance is 50 pF, the frequency error on that account would be less than 0.01%.

In a positive reactance oscillator, the crystal only needs to be in the intended response mode for the oscillator to satisfy a 0.5% or better frequency tolerance. That's because for any load capacitance the oscillation frequency is certain to be between the crystal's resonant and antiresonant frequencies.

Phase shifts that take place within the amplifier part of the oscillator will also affect frequency accuracy and stability. These phase shifts can normally be modeled as an "output capacitance" that, in the positive reactance oscillator, parallels C_{X2}. The predictability and constancy of this output capacitance over temperature and device sample will be the limiting factor in determining the tolerances that the circuit is capable of holding.

Drive Level

Drive level refers to the power dissipation in the crystal. There are two reasons for specifying it. One is that the parameters in the equivalent circuit are somewhat dependent on the drive level at which the crystal is calibrated. The other is that if the application circuit exceeds the test drive level by too much, the crystal may be damaged. Note that the terms "test drive level" and "rated drive level" both refer to the drive level at which the crystal is calibrated. Normally, in a microcontroller system, neither the frequency tolerances nor the power levels justify much concern for this specification. Some crystal manufacturers don't even require it for microprocessor crystals.

In a positive reactance oscillator, if one assumes the peak voltage across the crystal to be something in the neighborhood of V_{CC}, the power dissipation can be approximated as

$$P = 2R_1 [\pi f (C_L + C_0) V_{CC}]^2$$

This formula is derived in Appendix A. In a 5V system, P rarely evaluates to more than a milliwatt. Crystals with a standard 1 or 2 mW drive level rating can be used in most digital systems.

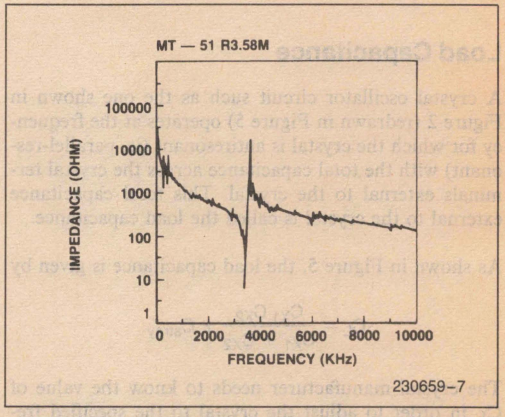


Figure 6. Ceramic Resonator Impedance vs. Frequency (Test Data Supplied by NTK Technical Ceramics)

CERAMIC RESONATORS

Ceramic resonators operate on the same basic principles as a quartz crsytal. Like quartz crsytals, they are piezoelectric, have a reactance versus frequency curve similar to a crystal's, and an equivalent circuit that looks just like a crystal's (with different parameter values, however).

The frequency tolerance of a ceramic resonator is about two orders of magnitude wider than a crystal's, but the ceramic is somewhat cheaper than a crystal. It may be noted for comparison that quartz crystals with relaxed tolerances cost about twice as much as ceramic resonators. For purposes of clocking a microcontroller, the frequency tolerance is often relatively noncritical, and the economic consideration becomes the dominant factor.

Figure 6 shows a graph of impedance magnitude versus frequency for a 3.58 MHz ceramic resonator. (Note that Figure 6 is a graph of $|Z_f|$ versus frequency, where

as Figure 3 is a graph of X_f versus frequency.) A number of spurious responses are apparent in Figure 6. The manufacturers state that spurious responses are more prevalent in the lower frequency resonators (kHz range) than in the higher frequency units (MHz range). For our purposes only the MHz range ceramics need to be considered.

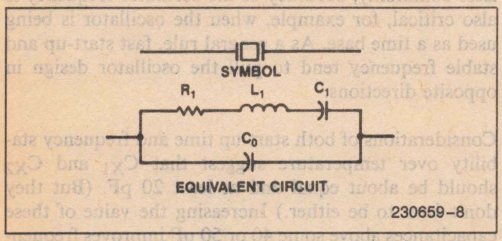


Figure 7. Ceramic Resonator: Symbol and Equivalent Circuit

Figure 7 shows the symbol and equivalent circuit for the ceramic resonator, both of which are the same as for the crystal. The parameters have different values, however, as listed in Table 2.

Table 2. Typical Ceramic Parameters

Frequency MHz	R_1 ohms	L_1 mH	C_1 pF	C_0 pF
3.58	7	0.113	19.6	140
6.0	8	0.094	8.3	60
8.0	7	0.092	4.6	40
11.0	10	0.057	3.9	30

Note that the motional arm of the ceramic resonator tends to have less resistance than the quartz crystal and also a vastly reduced L_1/C_1 ratio. This results in the motional arm having a Q (given by $(1/R_1)\sqrt{L_1/C_1}$) that is typically two orders of magnitude lower than that of a quartz crystal. The lower Q makes for a faster startup of the oscilaltor and for a less closely controlled frequency (meaning that circuitry external to the resonator will have more influence on the frequency than with a quartz crystal).

Another major difference is that the shunt capacitance of the ceramic resonator is an order of magnitude higher than C_0 of the quartz crystal and more dependent on the frequency of the resonator.

The implications of these differences are not all obvious, but some will be indicated in the section on Oscillator Calculations.

Specifications for Ceramic Resonators

Ceramic resonators are easier to specify than quartz crystals. All the vendor wants to know is the desired frequency and the chip you want it to work with. They'll supply the resonators, a circuit diagram showing the positions and values of other external components that may be required and a guarantee that the circuit will work properly at the specified frequency.

OSCILLATOR DESIGN CONSIDERATIONS

Designers of microcontroller systems have a number of options to choose from for clocking the system. The main decision is whether to use the "on-chip" oscillator or an external oscillator. If the choice is to use the on-chip oscillator, what kinds of external components are needed to make it operate as advertised? If the choice is to use an external oscillator, what type of oscillator should it be?

The decisions have to be based on both economic and technical requirements. In this section we'll discuss some of the factors that should be considered.

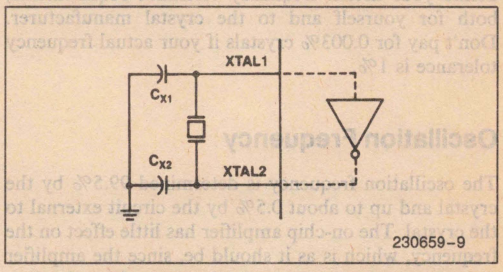


Figure 8. Using the "On-Chip" Oscillator

On-Chip Oscillators

In most cases, the on-chip amplifier with the appropriate external components provides the most economical solution to the clocking problem. Exceptions may arise in severe environments when frequency tolerances are tighter than about 0.01%.

The external components that need to be added are a positive reactance (normally a crystal or ceramic resonator) and the two capacitors C_{X1} and C_{X2}, as shown in Figure 8.

Crystal Specifications

Specifications for an appropriate crystal are not very critical, unless the frequency is. *Any* fundamental-mode crystal of medium or better quality can be used.

We are often asked what maximum crystal resistance should be specified. The best answer to this question is the lower the better, but use what's available. The crystal resistance will have some effect on start-up time and steady-state amplitude, but not so much that it can't be compensated for by appropriate selection of the capacitances C_{X1} and C_{X2}.

Similar questions are asked about specifications of load capacitance and shunt capacitance. The best advice we can give is to understand what these parameters mean and how they affect the operation of the circuit (that being the purpose of this Application Note), and then decide for yourself if such specifications are meaningful in your application or not. Normally, they're not, unless your frequency tolerances are tighter than about 0.1%.

Part of the problem is that crystal manufacturers are accustomed to talking "ppm" tolerances with radio engineers and simply won't take your order until you've filled out their list of specifications. It will help if you define your actual frequency tolerance requirements, both for yourself and to the crystal manufacturer. Don't pay for 0.003% crystals if your actual frequency tolerance is 1%.

Oscillation Frequency

The oscillation frequency is determined 99.5% by the crystal and up to about 0.5% by the circuit external to the crystal. The on-chip amplifier has little effect on the frequency, which is as it should be, since the amplifier parameters are temperature and process dependent.

The influence of the on-chip amplifier on the frequency is by means of its input and output (pin-to-ground) capacitances, which parallel C_{X1} and C_{X2}, and the XTAL1-to-XTAL2 (pin-to-pin) capacitance, which parallels the crystal. The input and pin-to-pin capacitances are about 7 pF each. Internal phase deviations from the nominal 180° can be modeled as an output capacitance of 25 to 30 pF. These deviations from the ideal have less effect in the positive reactance oscillator (with the inverting amplifer) than in a comparable series resonant oscillator (with the noninverting amplifier) for two reasons: first, the effect of the output capacitance is lessened, if not swamped, by the off-chip capacitor; secondly, the positive reactance oscillator is less sensitive, frequency-wise, to such phase errors.

Selection of C_{X1} and C_{X2}

Optimal values for the capacitors C_{X1} and C_{X2} depend on whether a quartz crystal or ceramic resonator is being used, and also on application-specific requirements on start-up time and frequency tolerance.

Start-up time is sometimes more critical in microcontroller systems than frequency stability, because of various reset and initialization requirements.

Less commonly, accuracy of the oscillator frequency is also critical, for example, when the oscillator is being used as a time base. As a general rule, fast start-up and stable frequency tend to pull the oscillator design in opposite directions.

Considerations of both start-up time and frequency stability over temperature suggest that C_{X1} and C_{X2} should be about equal and at least 20 pF. (But they don't *have* to be either.) Increasing the value of these capacitances above some 40 or 50 pF improves frequency stability. It also tends to increase the start-up time. There is a maximum value (several hundred pF, depending on the value of R_1 of the quartz or ceramic resonator) above which the oscillator won't start up at all.

If the on-chip amplifier is a simple inverter, such as in the 8051, the user can select values for C_{X1} and C_{X2} between some 20 and 100 pF, depending on whether start-up time or frequency stability is the more critical parameter in a specific application. If the on-chip amplifier is a Schmitt Trigger, such as in the 8048, smaller values of C_{X1} must be used (5 to 30 pF), in order to prevent the oscillator from running in a relaxation mode.

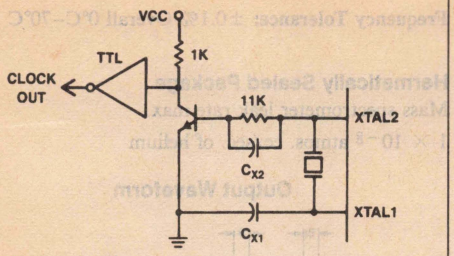
Later sections in this Application Note will discuss the effects of varying C_{X1} and C_{X2} (as well as other parameters), and will have more to say on their selection.

Placement of Components

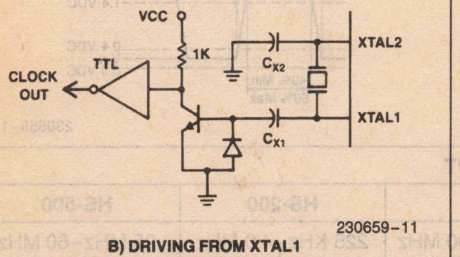
Noise glitches arriving at XTAL1 or XTAL2 pins at the wrong time can cause a miscount in the internal clock-generating circuitry. These kinds of glitches can be produced through capacitive coupling between the oscillator components and PCB traces carrying digital signals with fast rise and fall times. For this reason, the oscillator components should be mounted close to the chip and have short, direct traces to the XTAL1, XTAL2, and VSS pins.

Clocking Other Chips

There are times when it would be desirable to use the on-chip oscillator to clock other chips in the system.



A) DRIVING FROM XTAL2



B) DRIVING FROM XTAL1

Figure 9. Using the On-Chip Oscillator to Drive Other Chips

This can be done if an appropriate buffer is used. A TTL buffer puts too much load on the on-chip amplifier for reliable start-up. A CMOS buffer (such as the 74HC04) can be used, if it's fast enough and if its VIH and VIL specs are compatible with the available signal amplitudes. Circuits such as shown in Figure 9 might also be considered for these types of applications.

Clock-related signals are available at the TO pin in the MCS-48 products, at ALE in the MCS-48 and MCS-51 lines, and the iACX-96 controllers provide a CLKOUT signal.

External Oscillators

When technical requirements dictate the use of an external oscillator, the external drive requirements for the microcontroller, as published in the data sheet, must be carefully noted. The logic levels are not in general TTL-compatible. And each controller has its idiosyncracies in this regard. The 8048, for example, requires that both XTAL1 and XTAL2 be driven. The 8051 *can* be driven that way, but the data sheet suggest the simpler method of grounding XTAL1 and driving XTAL2. For this method, the driving source must be capable of sinking some current when XTAL2 is being driven low.

For the external oscillator itself, there are basically two choices: ready-made and home-grown.

TTL Crystal Clock Oscillator

The HS-100, HS-200, & HS-500 all-metal package series of oscillators are TTL compatible & fit a DIP layout. Standard electrical specifications are shown below. Variations are available for special applications.

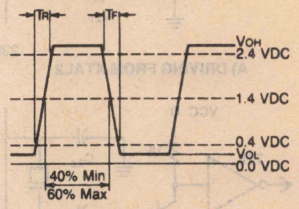
Frequency Range: HS-100—3.5 MHz to 30 MHz
HS-200—225 KHz to 3.5 MHz
HS-500—25 MHz to 60 MHz

Frequency Tolerance: ±0.1% Overall 0°C–70°C

Hermetically Sealed Package

Mass spectrometer leak rate max.
1×10^{-8} atmos. cc/sec. of helium

Output Waveform



230659-12

INPUT				
	HS-100		HS-200	HS-500
	3.5 MHz–20 MHz	20+ MHz–30 MHz	225 KHz–4.0 MHz	25 MHz–60 MHz
Supply Voltage (V_{CC})	5V ±10%	5V ±10%	5V ±10%	5V ±10%
Supply Current (I_{CC}) max.	30 mA	40 mA	85 mA	50 mA

OUTPUT				
	HS-100		HS-200	HS-500
	3.5 MHz–20 MHz	20+ MHz–30 MHz	225 KHz–4.0 MHz	25 MHz–60 MHz
V_{OH} (Logic "1")	+2.4V min.[1]	+2.7V min.[2]	+2.4V min.[1]	+2.7V min.[2]
V_{OL} (Logic "0")	+0.4V max.[3]	+0.5V max.[4]	+0.4V max.[3]	+0.5V max.[4]
Symmetry	60/40%[5]	60/40%[5]	55/45%[5]	60/40%[5]
T_R, T_F (Rise & Fall Time)	< 10 ns[6]	< 5 ns[6]	< 15 ns[6]	< 5 ns[6]
Output Short Circuit Current	18 mA min.	40 mA min.	18 mA min.	40 mA min.
Output Load	1 to 10 TTL Loads[7]	1 to 10 TTL Loads[8]	1 to 10 TTL Loads[7]	1 to 10 TTL Loads[8]

CONDITIONS

[1] I_O source = −400 μA max.
[2] I_O source = −1.0 mA max.
[3] I_O sink = 16.0 mA max.
[4] I_O sink = 20.00 mA max.
[5] V_O = 1.4V
[6] (0.4V to 2.4V)
[7] 1.6 mA per load
[8] 2.0 mA per load

Figure 10. Pre-Packaged Oscillator Data*

*Reprinted with the permission of © Midland-Ross Corporation 1982.

Prepackaged oscillators are available from most crystal manufacturers, and have the advantage that the system designer can treat the oscillator as a black box whose performance is guaranteed by people who carry many years of experience in designing and building oscillators. Figure 10 shows a typical data sheet for some prepackaged oscillators. Oscillators are also available with complementary outputs.

If the oscillator is to drive the microcontroller directly, one will want to make a careful comparison between the external drive requirements in the microcontroller data sheet and the oscillator's output logic levels and test conditions.

If oscillator stability is less critical than cost, the user may prefer to go with an in-house design. Not without some precautions, however.

It's easy to design oscillators that work. Almost all of them do work, even if the designer isn't too clear on why. The key point here is that *almost* all of them work. The problems begin when the system goes into production, and marginal units commence malfunctioning in the field. Most digital designers, after all, are not very adept at designing oscillators *for production.*

Oscillator design is somewhat of a black art, with the quality of the finished product being *very* dependent on the designer's experience and intuition. For that reason the most important consideration in any design is to have an adequate preproduction test program. Preproduction tests are discussed later in this Application Note. Here we will discuss some of the design options and take a look at some commonly used configurations.

Gate Oscillators versus Discrete Devices

Digital systems designers are understandably reluctant to get involved with discrete devices and their peculiarities (biasing techniques, etc.). Besides, the component count for these circuits tends to be quite a bit higher than what a digital designer is used to seeing for that amount of functionality. Nevertheless, if there are unusual requirements on the accuracy and stability of the clock frequency, it should be noted that discrete device oscillators can be tailored to suit the exact needs of the application and perfected to a level that would be difficult for a gate oscillator to approach.

In most cases, when an external oscillator is needed, the designer tends to rely on some form of a gate oscillator. A TTL inverter with a resistor connecting the output to the input makes a suitable inverting amplifier. The resistor holds the inverter in the transition region between logical high and low, so that at least for start-up purposes the inverter is a linear amplifier.

The feedback resistance has to be quite low, however, since it must conduct current sourced by the input pin without allowing the DC input voltage to get too far above the DC output voltage. For biasing purposes, the feedback resistance should not exceed a few k-ohms. But shunting the crystal with such a low resistance does not encourage start-up.

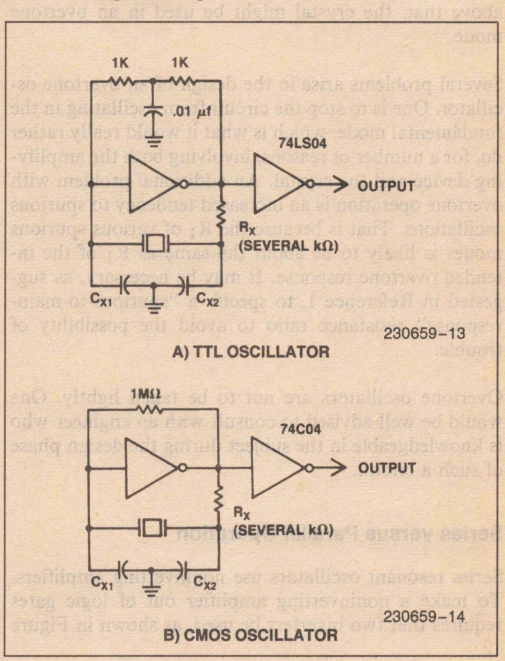


Figure 11. Commonly Used Gate Oscillators

Consequently, the configuration in Figure 11A might be suggested. By breaking R_f into two parts and AC-grounding the midpoint, one achieves the DC feedback required to hold the inverter in its active region, but without the negative signal feedback that is in effect telling the circuit *not* to oscillate. However, this biasing scheme will increase the start-up time, and relaxation-type oscillations are also possible.

A CMOS inverter, such as the 74HC04, might work better in this application, since a larger R_f can be used to hold the inverter in its linear region.

Logic gates tend to have a fairly low output resistance, which destabilizes the oscillator. For that reason a resistor Rx is often added to the feedback network, as shown in Figures 11A and B. At higher frequencies a 20 or 30 pF capacitor is sometimes used in the Rx position, to compensate for some of the internal propagation delay.

Reference 1 contains an excellent discussion of gate oscillators, and a number of design examples.

Fundamental versus Overtone Operation

It's easier to design an oscillator circuit to operate in the resonator's fundamental response mode than to design one for overtone operation. A quartz crystal whose fundamental response mode covers the desired frequency can be obtained up to some 30 MHz. For frequencies above that, the crystal might be used in an overtone mode.

Several problems arise in the design of an overtone oscillator. One is to stop the circuit from oscillating in the fundamental mode, which is what it would really rather do, for a number of reasons, involving both the amplifying device and the crystal. An additional problem with overtone operation is an increased tendency to spurious oscillations. That is because the R_1 of various spurious modes is likely to be about the same as R_1 of the intended overtone response. It may be necessary, as suggested in Reference 1, to specify a "spurious-to-main-response" resistance ratio to avoid the possibility of trouble.

Overtone oscillators are not to be taken lightly. One would be well advised to consult with an engineer who is knowledgeable in the subject during the design phase of such a circuit.

Series versus Parallel Operation

Series resonant oscillators use noninverting amplifiers. To make a noninverting amplifier out of logic gates requires that two inverters be used, as shown in Figure 12.

This type of circuit tends to be inaccurate and unstable in frequency over variations in temperature and V_{CC}. It has a tendency to oscillate at overtones, and to oscillate through C_0 of the crystal or some stray capacitance rather than as controlled by the mechanical resonance of the crystal.

The demon in series resonant oscillators is the phase shift in the amplifier. The series resonant oscillator wants more than just a "noninverting" amplifier—it wants a *zero phase-shift* amplifier. Multistage noninverting amplifiers tend to have a considerably lagging phase shift, such that the crystal reactance must be capacitive in order to bring the total phase shift around the feedback loop back up to 0. In this mode, a "12 MHz" crystal may be running at 8 or 9 MHz. One can put a capacitor in series with the crystal to relieve the crystal of having to produce all of the required phase shift, and bring the oscillation frequency closer to fs. However, to further complicate the situation, the amplifier's phase shift is strongly dependent on frequency, temperature, VCC, and device sample.

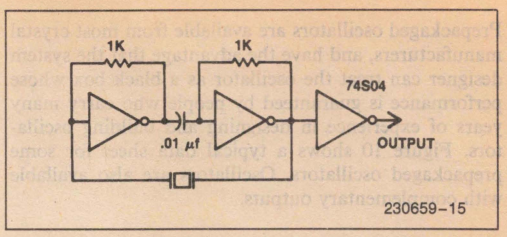


Figure 12. "Series Resonant" Gate Oscillator

Positive reactance oscillators ("parallel resonant") use inverting amplifiers. A single logic inverter can be used for the amplifier, as in Figure 11. The amplifier's phase shift is less critical, compared to a series resonant circuit, and since only one inverter is involved there's less phase error anyway. The oscillation frequency is effectively bounded by the resonant and antiresonant frequencies of the crystal itself. In addition, the feedback network includes capacitors that parallel the input and output terminals of the amplifier, thus reducing the effect of unpredictable capacitances at these points.

MORE ABOUT USING THE "ON-CHIP" OSCILLATORS

In this section we will describe the on-chip inverters on selected microcontrollers in some detail, and discuss criteria for selecting components to work with them. Future data sheets will supplement this discussion with updates and information pertinent to the use of each chip's oscillator circuitry.

Oscillator Calculations

Oscillator design, though aided by theory, is still largely an empirical exercise. The circuit is inherently nonlinear, and the normal analysis parameters vary with instantaneous voltage. In addition, when dealing with the on-chip circuitry, we have FETs being used as resistors, resistors being used as interconnects, distributed delays, input protection devices, parasitic junctions, and processing variations.

Consequently, oscillator calculations are never very precise. They can be useful, however, if they will at least indicate the effects of *variations* in the circuit parameters on start-up time, oscillation frequency, and steady-state amplitude. Start-up time, for example, can be taken as an indication of start-up reliability. If preproduction tests indicate a possible start-up problem, a relatively inexperienced designer can at least be made aware of what parameter may be causing the marginality, and what direction to go in to fix it.

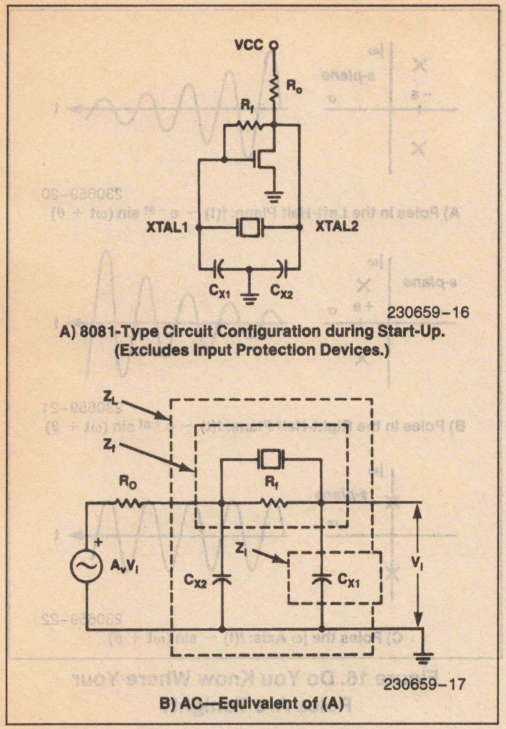


Figure 13. Oscillator Circuit Model Used in Start-Up Calculations

The analysis used here is mathematically straightforward but algebraically intractable. That means it's relatively easy to understand and program into a computer, but it will not yield a neat formula that gives, say, steady-state amplitude as a function of this or that list of parameters. A listing of a BASIC program that implements the analysis will be found in Appendix II.

When the circuit is first powered up, and before the oscillations have commenced (and if the oscillations *fail* to commence), the oscillator can be treated as a small signal linear amplifier with feedback. In that case, standard small-signal analysis techniques can be used to determine start-up characteristics. The circuit model used in this analysis is shown in Figure 13.

The circuit approximates that there are no high-frequency effects within the amplifier itslef, such that its high-frequency behavior is dominated by the load impedance Z_L. This is a reasonable approximation for single-stage amplifiers of the type used in 8051-type devices. Then the gain of the amplifier as a function of frequency is

$$A = \frac{A_V Z_L}{Z_L + R_O}$$

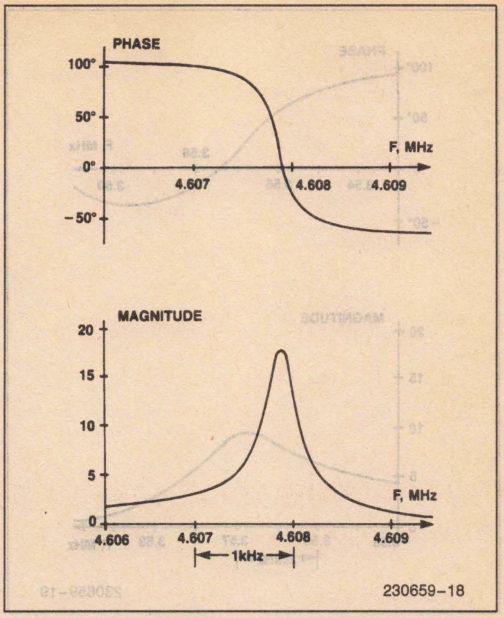


Figure 14. Loop Gain versus Frequency (4.608 MHz Crystal)

The gain of the feedback network is

$$\beta = \frac{Z_i}{Z_i + Z_f}$$

And the loop gain is

$$\beta A = \frac{Z_i}{Z_i + Z_f} \times \frac{A_V Z_L}{Z_L + R_O}$$

The impedances Z_L, Z_f, and Z_i are defined in Figure 13B.

Figure 14 shows the way the loop gain thus calculated (using typical 8051-type parameters and a 4.608 MHz crystal) varies with frequency. The frequency of interest is the one for which the phase of the loop gain is zero. The accepted criterion for start-up is that the magnitude of the loop gain must exceed unity at this frequency. This is the frequency at which the circuit is in resonance. It corresponds very closely with the antiresonant frequency of the motional arm of the crystal in parallel with C_L.

Figure 15 shows the way the loop gain varies with frequency when the parameters of a 3.58 MHz ceramic resonator are used in place of a crystal (the amplifier parameters being typical 8051, as in Figure 14). Note the different frequency scales.

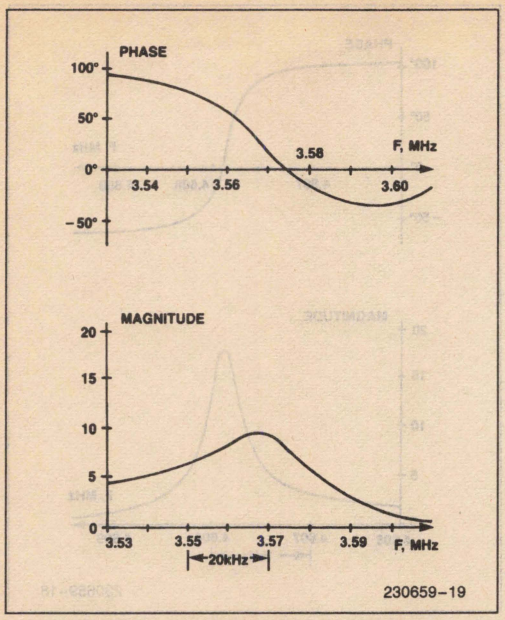


Figure 15. Loop Gain versus Frequency (3.58 MHz Ceramic)

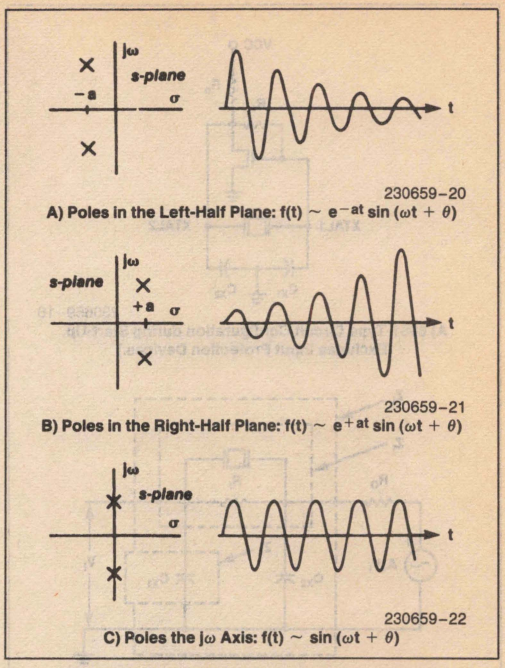


Figure 16. Do You Know Where Your Poles Are Tonight?

Start-Up Characteristics

It is common, in studies of feedback systems, to examine the behavior of the closed loop gain as a function of complex frequency $s = \sigma + j\omega$; specifically, to determine the location of its poles in the complex plane. A pole is a point on the complex plane where the gain function goes to infinity. Knowledge of its location can be used to predict the response of the system to an input disturbance.

The way that the response function depends on the location of the poles is shown in Figure 16. Poles in the left-half plane cause the response function to take the form of a damped sinusoid. Poles in the right-half plane cause the response function to take the form of an exponentially growing sinusoid. In general,

$$v(t) \sim e^{at} \sin(\omega t + \theta)$$

where *a* is the real part of the pole frequency. Thus if the pole is in the right-half plane, a is positive and the sinusoid grows. If the pole is in the left-half plane, a is negative and the sinusoid is damped.

The same type of analysis can usefully be applied to oscillators. In this case, however, rather than trying to ensure that the poles are in the left-half plane, we would seek to ensure that they're in the *right*-half plane. An exponentially growing sinusoid is exactly what is wanted from an oscillator that has just been powered up.

The gain function of interest in oscillators is $1/(1 - \beta A)$. Its poles are at the complex frequencies where $\beta A = 1\angle 0°$, because that value of βA causes the gain function to go to infinity. The oscillator will start up if the real part of the pole frequency is positive. More importantly, the *rate* at which it starts up is indicated by how *much* greater than 0 the real part of the pole frequency is.

The circuit in Figure 13B can be used to find the pole frequencies of the oscillator gain function. All that needs to be done is evaluate the impedances at complex frequencies $\sigma + j\omega$ rather than just at ω, and find the value of $\sigma + j\omega$ for which $\beta A = 1\angle 0°$. The larger that value of σ is, the faster the oscillator will start up.

Of course, other things besides pole frequencies, things like the VCC rise time, are at work in determining the start-up time. But to the extend that the pole frequencies *do* affect start-up time, we can obtain results like those in Figures 17 and 18.

To obtain these figures the pole frequencies were computed for various values of capacitance C_X from XTAL1 and XTAL2 to ground (thus $C_{X1} = C_{X2} = C_X$). Then a "time constant" for start-up was calculated as $T_s = \frac{1}{\sigma}$ where σ is the real part of the pole frequency (rad/sec), and this time constant is plotted versus C_X.

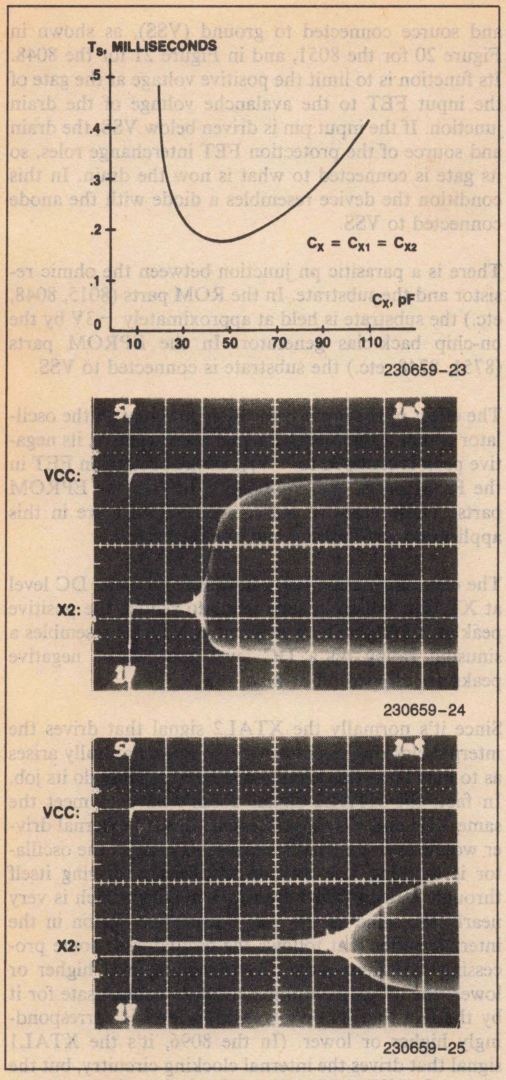


Figure 17. Oscillator Start-Up (4.608 MHz Crystal from Standard Crystal Corp.)

A short time constant means faster start-up. A long time constant means slow start-up. Observations of actual start-ups are shown in the figures. Figure 17 is for a typical 8051 with a 4.608 MHz crystal supplied by Standard Crystal Corp., and Figure 18 is for a typical 8051 with a 3.58 MHz ceramic resonator supplied by NTK Technical Ceramics, Ltd.

It can be seen in Figure 17 that, for this crystal, values of C_X between 30 and 50 pF minimize start-up time, but that the exact value in this range is not particularly important, even if the start-up time itself is critical.

As previously mentioned, start-up time can be taken as an indication of start-up reliability. Start-up problems are normally associated with C_{X1} and C_{X2} being too small or too large for a given resonator. If the parameters of the resonator are known, curves such as in Figure 17 or 18 can be generated to define acceptable ranges of values for these capacitors.

As the oscillations grow in amplitude, they reach a level at which they undergo severe clipping within the amplifier, in effect reducing the amplifier gain. As the amplifier gain decreases, the poles move towards the jω axis. In steady-state, the poles are on the jω axis and the amplitude of the oscillations is constant.

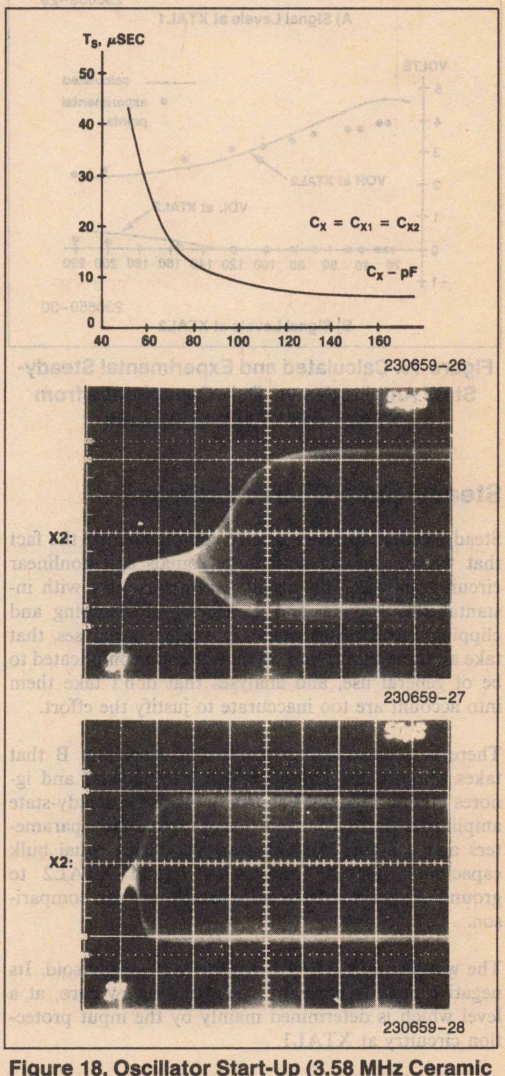


Figure 18. Oscillator Start-Up (3.58 MHz Ceramic Resonator from NTK Technical Ceramics)

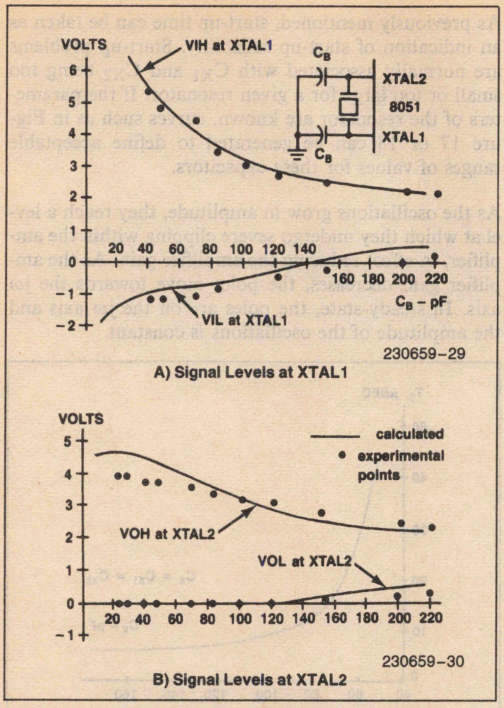


Figure 19. Calculated and Experimental Steady-State Amplitudes vs. Bulk Capacitance from XTAL1 and XTAL2 to Ground

Steady-State Characteristics

Steady-state analysis is greatly complicated by the fact that we are dealing with large signals and nonlinear circuit response. The circuit parameters vary with instantaneous voltage, and a number of clamping and clipping mechanisms come into play. Analyses that take all these things into account are too complicated to be of general use, and analyses that don't take them into account are too inaccurate to justify the effort.

There is a steady-state analysis in Appendix B that takes some of the complications into account and ignores others. Figure 19 shows the way the steady-state amplitudes thus calculated (using typical 8051 parameters and a 4.608 MHz crystal) vary with equal bulk capacitance placed from XTAL1 and XTAL2 to ground. Experimental results are shown for comparison.

The waveform at XTAL1 is a fairly clean sinusoid. Its negative peak is normally somewhat below zero, at a level which is determined mainly by the input protection circuitry at XTAL1.

The input protection circuitry consists of an ohmic resistor and an enhancement-mode FET with the gate and source connected to ground (VSS), as shown in Figure 20 for the 8051, and in Figure 21 for the 8048. Its function is to limit the positive voltage at the gate of the input FET to the avalanche voltage of the drain junction. If the input pin is driven below VSS, the drain and source of the protection FET interchange roles, so its gate is connected to what is now the drain. In this condition the device resembles a diode with the anode connected to VSS.

There is a parasitic pn junction between the ohmic resistor and the substrate. In the ROM parts (8015, 8048, etc.) the substrate is held at approximately −3V by the on-chip back-bias generator. In the EPROM parts (8751, 8748, etc.) the substrate is connected to VSS.

The effect of the input protection circuitry on the oscillator is that if the XTAL1 signal goes negative, its negative peak is clamped to $-V_{DS}$ of the protection FET in the ROM parts, and to about −0.5V in the EPROM parts. These negative voltages on XTAL1 are in this application self-limiting and nondestructive.

The clamping action does, however, raise the DC level at XTAL1, which in turn tends to reduce the positive peak at XTAL2. The waveform at XTAL2 resembles a sinusoid riding on a DC level, and whose negative peaks are clipped off at zero.

Since it's normally the XTAL2 signal that drives the internal clocking circuitry, the question naturally arises as to how large this signal must be to reliably do its job. In fact, the XTAL2 signal doesn't have to meet the same VIH and VIL specifications that an external driver would have to. That's because as long as the oscillator is working, the on-chip amplifier is driving itself through its own 0-to-1 transition region, which is very nearly the same as the 0-to-1 transition region in the internal buffer that follows the oscillator. If some processing variations move the transition level higher or lower, the on-chip amplifier tends to compensate for it by the fact that its own transition level is correspondingly higher or lower. (In the 8096, it's the XTAL1 signal that drives the internal clocking circuitry, but the same concept applies.)

The main concern about the XTAL2 signal amplitude is an indication of the general health of the oscillator. An amplitude of less than about 2.5V peak-to-peak indicates that start-up problems could develop in some units (with low gain) with some crystals (with high R_1). The remedy is to either adjust the values of C_{X1} and/or C_{X2} or use a crystal with a lower R_1.

The amplitudes at XTAL1 and XTAL2 can be adjusted by changing the ratio of the capacitors from XTAL1 and XTAL2 to ground. Increasing the XTAL2 capacitance, for example, decreases the amplitude at XTAL2 and increases the amplitude at XTAL1 by about the same amount. Decreasing both caps increases both amplitudes.

Pin Capacitance

Internal pin-to-ground and pin-to-pin capacitances at XTAL1 and XTAL2 will have some effect on the oscillator. These capacitances are normally taken to be in the range of 5 to 10 pF, but they are extremely difficult to evaluate. Any measurement of one such capacitance will necessarily include effects from the others. One advantage of the positive reactance oscillator is that the pin-to-ground capacitances are paralleled by external bulk capacitors, so a precise determination of their value is unnecessary. We would suggest that there is little justification for more precision than to assign them a value of 7 pF (XTAL1-to-ground and XTAL1-to-XTAL2). This value is probably not in error by more than 3 or 4 pF.

The XTAL2-to-ground capacitance is not entirely "pin capacitance," but more like an "equivalent output capacitance" of some 25 to 30 pF, having to include the effect of internal phase delays. This value will vary to some extent with temperature, processing, and frequency.

MCS®-51 Oscillator

The on-chip amplifier on the HMOS MCS-51 family is shown in Figure 20. The drain load and feedback "resistors" are seen to be field-effect transistors. The drain load FET, R_D, is typically equivalent to about 1K to 3 K-ohms. As an amplifier, the low frequency voltage gain is normally between -10 and -20, and the output resistance is effectively R_D.

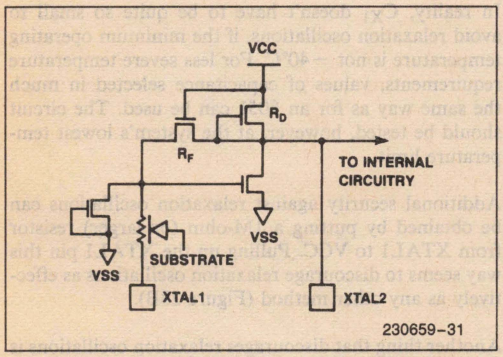


Figure 20. MCS®-51 Oscillator Amplifier

The 80151 oscillator is normally used with equal bulk capacitors placed externally from XTAL1 to ground and from XTAL2 to ground. To determine a reasonable value of capacitance to use in these positions, given a crystal of ceramic resonator of known parameters, one can use the BASIC analysis in Appendix II to generate curves such as in Figures 17 and 18. This procedure will define a range of values that will minimize start-up time. We don't suggest that smaller values be used than those which minimize start-up time. Larger values than those can be used in applications where increased frequency stability is desired, at some sacrifice in start-up time.

Standard Crystal Corp. (Reference 8) studied the use of their crystals with the MCS-51 family using skew sample supplied by Intel. They suggest putting 30 pF capacitors from XTAL1 and XTAL2 to ground, if the crystal is specified as described in Reference 8. They noted that in that configuration and with crystals thus specified, the frequency accuracy was $\pm 0.01\%$ and the frequency stability was $\pm 0.005\%$, and that a frequency accuracy of $\pm 0.005\%$ could be obtained by substituting a 25 pF fixed cap in parallel with a 5–20 pF trimmer for one of the 30 pF caps.

MCS-51 skew samples have also been supplied to a number of ceramic resonator manufacturers for characterization with their products. These companies should be contacted for application information on their products. In general, however, ceramics tend to want somewhat larger values for C_{X1} and C_{X2} than quartz crystals do. As shown in Figure 18, they start up a lot faster that way.

In some application the actual frequency tolerance required is only 1% or so, the user being concerned mainly that the circuit *will* oscillate. In that case, C_{X1} and C_{X2} can be selected rather freely in the range of 20 to 80 pF.

As you can see, "best" values for these components and their tolerances are strongly dependent on the application and its requirements. In any case, their suitability should be verified by environmental testing before the design is submitted to production.

MCS®-48 Oscillator

The NMOS and HMOS MCS-48 oscillator is shown in Figure 21. It differs from the 8051 in that its inverting

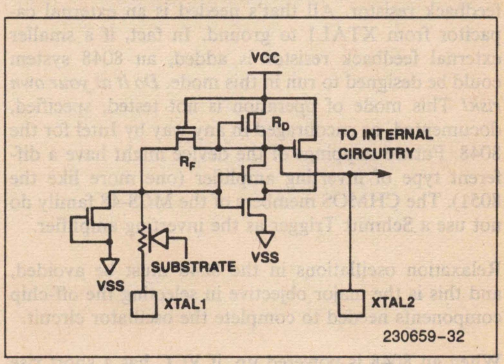


Figure 21. MCS®-48 Oscillator Amplifier

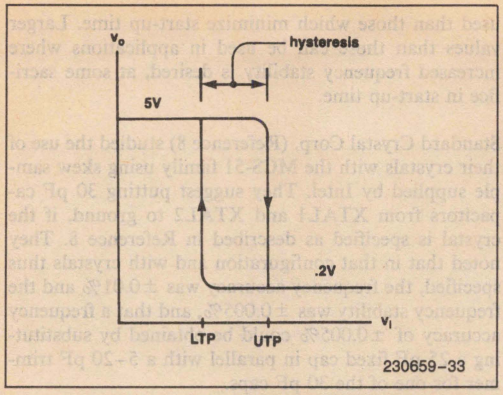


Figure 22. Schmitt Trigger Characteristic

amplifier is a Schmitt Trigger. This configuration was chosen to prevent crosstalk from the TO pin, which is adjacent to the XTAL1 pin.

All Schmitt Trigger circuits exhibit a hysteresis effect, as shown in Figure 22. The hysteresis is what makes it less sensitive to noise. The same hysteresis allows any Schmitt Trigger to be used as a relaxation oscillator. All you have to do is connect a resistor from output to input, and a capacitor from input to ground, and the circuit oscillates in a relaxation mode as follows.

If the Schmitt Trigger output is at a logic high, the capacitor commences charging through the feedback resistor. When the capacitor voltage reaches the upper trigger point (UTP), the Schmitt Trigger output switches to a logic low and the capacitor commences discharging through the same resistor. When the capacitor voltage reaches the lower trigger point (LTP), the Schmitt Trigger output switches to a logic high again, and the sequence repeats. The oscillation frequency is determined by the RC time constant and the hysteresis voltage, UTP-LTP.

The 8048 can oscillate in this mode. It has an internal feedback resistor. All that's needed is an external capacitor from XTAL1 to ground. In fact, if a smaller external feedback resistor is added, an 8048 system could be designed to run in this mode. *Do it at your own risk!* This mode of operation is not tested, specified, documented, or encouraged in any way by Intel for the 8048. Future steppings of the device might have a different type of inverting amplifier (one more like the 8051). The CHMOS members of the MCS-48 family do not use a Schmitt Trigger as the inverting amplifier.

Relaxation oscillations in the 8048 must be avoided, and this is the major objective in selecting the off-chip components needed to complete the oscillator circuit.

When an 8048 is powered up, if VCC has a short rise time, the relaxation mode starts first. The frequency is normally about 50 KHz. The resonator mode builds more slowly, but it eventually takes over and dominates the operation of the cirucit. This is shown in Figure 23A.

Due to processing variations, some units seem to have a harder time coming out of the relaxation mode, particularly at low temperatures. In some cases the resonator oscillations may fail entirely, and leave the device in the relaxation mode. Most units will stick in the relaxation mode at any temperature if C_{X1} is larger than about 50 pF. Therefore, C_{X1} should be chosen with some care, particularly if the system must operate at lower temperatures.

One method that has proven effective in all units to $-40°C$ is to put 5 pF from XTAL1 to ground and 20 pF from XTAL2 to ground. Unfortunately, while this method does discourage the relaxation mode, it is not an optimal choice for the resonator mode. For one thing, it does not swamp the pin capacitance. Also, it makes for a rather high signal level at XTAL1 (8 or 9 volts peak-to-peak).

The question arises as to whether that level of signal at XTLA1 might damage the chip. Not to worry. The negative peaks are self-limiting and nondestructive. The positive peaks could conceivably damage the oxide, but in fact, NMOS chips (eg, 8048) and HMOS chips (eg, 8048H) are tested to a much higher voltage than that. The technology trend, of course, is to thinner oxides, as the devices shrink in size. For an extra margin of safety, the HMOS II chips (eg, 8048AH) have an internal diode clamp at XTAL1 to VCC.

In reality, C_{X1} doesn't have to be quite so small to avoid relaxation oscillations, if the minimum operating temperature is not $-40°C$. For less severe temperature requirements, values of capacitance selected in much the same way as for an 8051 can be used. The circuit should be tested, however, at the system's lowest temperature limit.

Additional security against relaxation oscillations can be obtained by putting a 1M-ohm (or larger) resistor from XTAL1 to VCC. Pulling up the XTAL1 pin this way seems to discourage relaxation oscillations as effectively as any other method (Figure 23B).

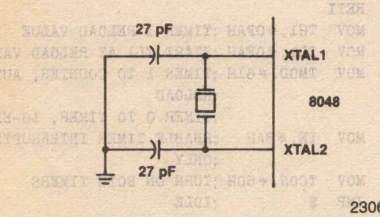
Another thing that discourages relaxation oscillations is low VCC. The resonator mode, on the other hand is much less sensitive to VCC. Thus if VCC comes up relatively slowly (several milliseconds rise time), the resonator mode is normally up and running before the relaxation mode starts (in fact, before VCC has even reached operating specs). This is shown in Figure 23C.

A secondary effect of the hysteresis is a shift in the oscillation frequency. At low frequencies, the output signal from an inverter without hysteresis leads (or lags) the input by 180 degrees. The hysteresis in a Schmitt Trigger, however, causes the output to lead the

input by less than 180 degrees (or lag by more than 180 degrees), by an amount that depends on the signal amplitude, as shown in Figure 24. At higher frequencies, there are additional phase shifts due to the various reactances in the circuit, but the phase shift due to the hysteresis is still present. Since the total phase shift in the oscillator's loop gain is necessarily 0 or 360 degrees, it is apparent that as the oscillations build up, the frequency has to change to allow the reactances to compensate for the hysteresis. In normal operation, this additional phase shift due to hysteresis does not exceed a few degrees, and the resulting frequency shift is negligible.

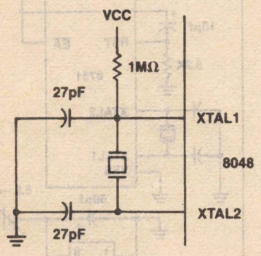
Kyocera, a ceramic resonator manufacturer, studied the use of some of their resonators (at 6.0 MHz, 8.0 MHz, and 11.0 MHz) with the 8049H. Their conclusion as to the value of capacitance to use at XTAL1 and XTAL2 was that 33 pF is appropriate at all three frequencies. One should probably follow the manufacturer's recommendations in this matter, since they will guarantee operation.

Whether one should accept these recommendations and guarantees without further testing is, however, another matter. Not all users have found the recommendations to be without occasional problems. If you run into diffi-



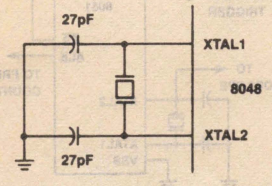
230659-34

A) When VCC Comes Up Fast, Relaxation Oscillations Start First. But Then the Crystal Takes Over.



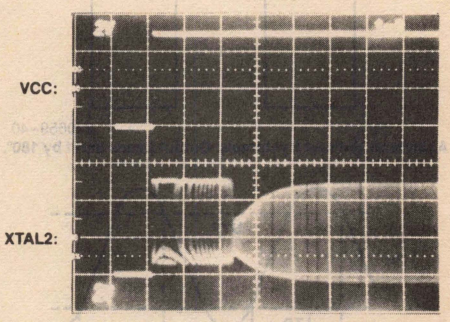
230659-35

B) Weak Pullup (1 MΩ or More) on XTAL1 Discourages Relaxation Mode.

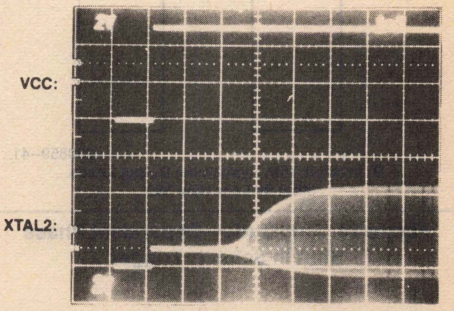


230659-36

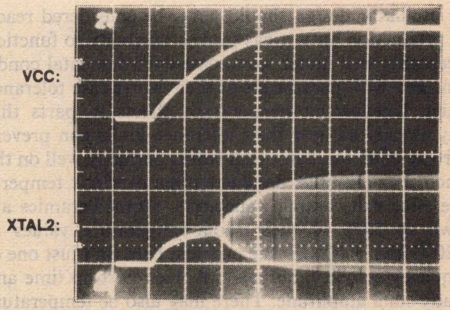
C) No Relaxation Oscillations When VCC Comes Up More Slowly.



230659-37



230659-38



230659-39

Figure 23. Relaxation Oscillations in the 8048

culties using their recommendations, both Intel and the ceramic resonator manufacturer want to know about it. It is to their interest, and ours, that such problems be resolved.

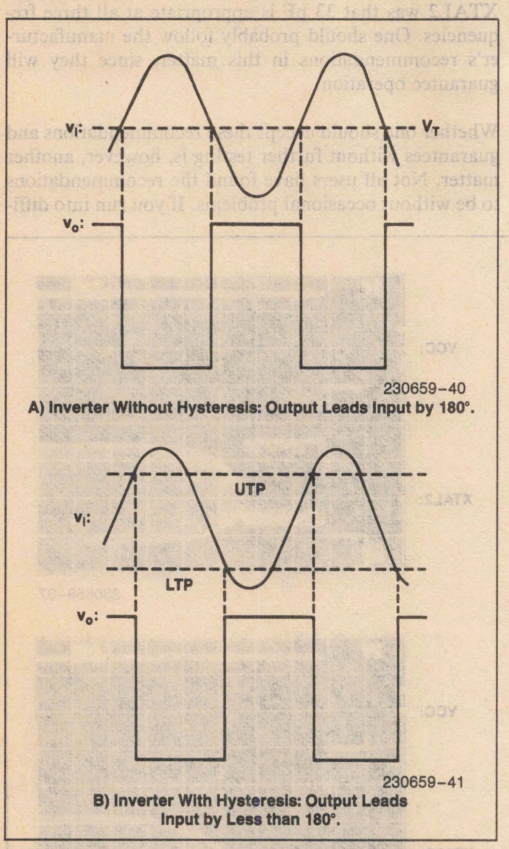


Figure 24. Amplitude—Dependent Phase Shift in Schmitt Trigger

Preproduction Tests

An oscillator design should never be considered ready for production until it has proven its ability to function acceptably well under worst-case environmental conditions and with parameters at their worst-case tolerance limits. Unexpected temperature effects in parts that may already be near their tolerance limits can prevent start-up of an oscillator that works perfectly well on the bench. For example, designers often overlook temperature effects in ceramic capacitors. (Some ceramics are down to 50% of their room-temperature values at −20°C and +60°C). The problem here isn't just one of frequency stability, but also involves start-up time and steady-state amplitude. There may also be temperature effects in the resonator and amplifier.

It will be helpful to build a test jig that will allow the oscillator circuit to be tested independently of the rest of the system. Both start-up and steady-state characteristics should be tested. Figure 25 shows the circuit that

A) Software for Oscillator Test

```
SOURCE
ORG 0000 H
        JMP  START
ORG 000B H              ;TIMER 0 INTERRUPT:
        CPL  T1         ;  TOGGLE T1
        RETI
ORG 0001BH              ;TIMER 1 INTERRUPT:
        CPL  P1.1       ;  TOGGLE CRO TRIGGER
        DJNZ P2,$       ;  DELAY
        CPL  P1.0       ;  TOGGLE VCC CONTROL
        RETI
START:  MOV  TH1,#0FAH ;TIMER 1 RELOAD VALUE
        MOV  TL1,#0FAH ;START TL1 AT RELOAD VALUE
        MOV  TMOD,#61H ;TIMER 1 TO COUNTER, AUTO
                       ;RELOAD
                       ;TIMER 0 TO TIMER, 16-BIT
        MOV  IE,#BAH   ;ENABLE TIMER INTERRUPTS
                       ;ONLY
        MOV  TCON,#50H ;TURN ON BOTH TIMERS
        JMP  $         ;IDLE
END
```

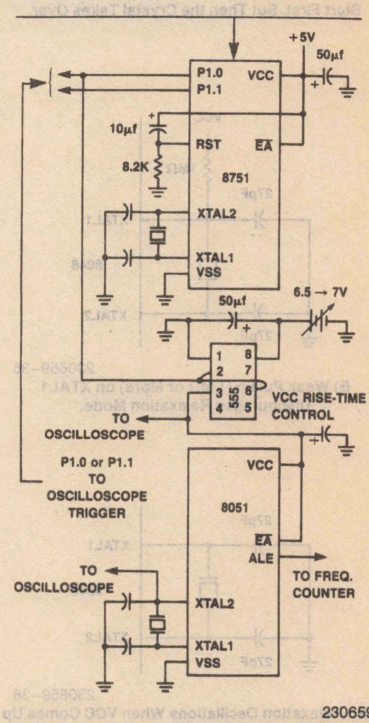


Figure 25. Oscillator Test Circuit and Software

was used to obtain the oscillator start-up photographs in this Application Note. This circuit or a modified version of it would make a convenient test vehicle. The oscillator and its relevant components can be physically separated from the control circuitry, and placed in a temperature chamber.

Start-up should be observed under a variety of conditions, including low VCC and using slow and fast VCC rise times. The oscillator should not be reluctant to start up even when VCC is below its spec value for the rest of the chip. (The rest of the chip may not function, but the oscillator should work.) It should also be verified that start-up occurs when the resonator has more than its upper tolerance limit of series resistance. (Put some resistance in series with the resonator for this test.) The bulk capacitors from XTAL1 and XTAL2 to ground should also be varied to their tolerance limits.

The same circuit, with appropriate changes in the software to lengthen the "on" time, can be used to test the steady-state characteristics of the oscillator, specifically the frequency, frequency stability, and amplitudes at XTAL1 and XTAL2.

As previously noted, the voltage swings at these pins are not critical, but they should be checked at the system's temperature limits to ensure that they are in good health. Observing these signals necessarily changes them somewhat. Observing the signal at XTAL2 requires that the capacitor at that pin be reduced to account for the oscilloscope probe capacitance. Observing the signal at XTAL1 requires the same consideration, plus a blocking capacitor (switch the oscilloscope input to AC), so as to not disturb the DC level at that pin. Alternatively, a MOSFET buffer such as the one shown in Figure 26 can be used. It should be verified by direct measurement that the ground clip on the scope probe is ohmically connected to the scope chassis (probes are incredibly fragile in this respect), and the observations should be made with the ground clip on the VSS pin, or very close to it. If the probe shield isn't operational and in use, the observations are worthless.

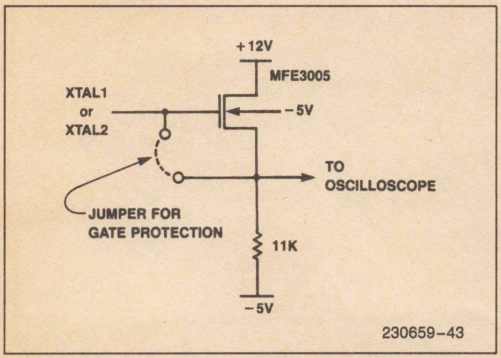


Figure 26. MOSFET Buffer for Observing Oscillator Signals

Frequency checks should be made with only the oscillator circuitry connected to XTAL1 and XTAL2. The ALE frequency can be counted, and the oscillator frequency derived from that. In systems where the frequency tolerance is only "nominal," the frequency should still be checked to ascertain that the oscillator isn't running in a spurious resonance or relaxation mode. Switching VCC off and on again repeatedly will help reveal a tendency to go into unwanted modes of oscillation.

The operation of the oscillator should then be verified under actual system running conditions. By this stage one will be able to have some confidence that the basic selection of components for the oscillator itself is suitable, so if the oscillator appears to malfunction in the system the fault is not in the selection of these components.

Troubleshooting Oscillator Problems

The first thing to consider in case of difficulty is that between the test jig and the actual application there may be significant differences in stray capacitances, particularly if the actual application is on a multi-layer board.

Noise glitches, that aren't present in the test jig but are in the application board, are another possibility. Capacitive coupling between the oscillator circuitry and other signal has already been mentioned as a source of miscounts in the internal clocking circuitry. Inductive coupling is also possible, if there are strong currents nearby. These problems are a function of the PCB layout.

Surrounding the oscillator components with "quiet" traces (VCC and ground, for example) will alleviate capacitive coupling to signals that have fast transition times. To minimize inductive coupling, the PCB layout should minimize the areas of the loops formed by the oscillator components. These are the loops that should be checked:

XTAL1 through the resonator to XTAL2;
XTAL1 through C_{X1} to the VSS pin;
XTAL2 through C_{X2} to the VSS pin.

It is not unusual to find that the grounded ends of C_{X1} and C_{X2} eventually connect up to the VSS pin only after looping around the farthest ends of the board. Not good.

Finally, it should not be overlooked that software problems sometimes imitate the symptoms of a slow-starting oscillator or incorrect frequency. Never underestimate the perversity of a software problem.

REFERENCES

1. Frerking, M. E., *Crystal Oscillator Design and Temperature Compensation,* Van Nostrand Reinhold, 1978.

2. Bottom, V., "The Crystal Unit as a Circuit Component," Ch. 7, *Introduction to Quartz Crystal Unit Design,* Van Nostrand Reinhold, 1982.

3. Parzen, B., *Design of Crystal and Other Harmonic Oscillators,* John Wiley & Sons, 1983.

4. Holmbeck, J. D., "Frequency Tolerance Limitations with Logic Gate Clock Oscillators, *31st Annual Frequency Control Symposium,* June, 1977.

5. Roberge, J. K., "Nonlinear Systems," Ch. 6, *Operational Amplifiers: Theory and Practice,* Wiley, 1975.

6. Eaton, S. S. *Timekeeping Advances Through COS/MOS Technology,* RCA Application Note ICAN-6086.

7. Eaton, S. S., *Micropower Crystal-Controlled Oscillator Design Using RCA COS/MOS Inverters,* RCA Application Note ICAN-6539.

8. Fisher, J. B., *Crystal Specifications for the Intel 8031/8051/8751 Microcontrollers,* Standard Crystal Corp. Design Data Note #2F.

9. Murata Mfg. Co., Ltd., *Ceramic Resonator "Ceralock" Application Manual.*

10. Kyoto Ceramic Co., Ltd., *Adaptability Test Between Intel 8049H and Kyocera Ceramic Resonators.*

11. Kyoto Ceramic Co., Ltd., *Technical Data on Ceramic Resonator Model KBR-6.0M, KBR-8.0M, KBR-11.0M Application for 8051 (Intel).*

12. NTK Technical Ceramic Division, NGK Spark Plug Co., Ltd., *NTKK Ceramic Resonator Manual.*

APPENDIX A
QUARTZ AND CERAMIC RESONATOR FORMULAS

Based on the equivalent circuit of the crystal, the impedance of the crystal is

$$Z_{XTAL} = \frac{(R_1 + j\omega L_1 + 1/j\omega C_1)\,(1/j\omega C_0)}{R_1 + j\omega L_1 + 1/j\omega C_1 + 1/j\omega C_0}$$

After some algebraic manipulation, this calculation can be written in the form

$$Z_{XTAL} = \frac{1}{j\omega(C_1 + C_0)} \bullet \frac{1 - \omega^2 L_1 C_1 + j\omega R_1 C_1}{1 - \omega^2 L_1 C_T + j\omega R_1 C_T}$$

where C_T is the capacitance of C_1 in series with C_0:

$$C_T = \frac{C_1 C_0}{C_1 + C_0}$$

The impedance of the crystal in parallel with an external load capacitance C_L is the same expression, but with $C_0 + C_L$ substituted for C_0:

$$Z_{XTAL}\,\|_{CL} = \frac{1}{j\omega(C_1 + C_0 + C_L)} \bullet \frac{1 - \omega^2 L_1 C_1 + j\omega R_1 C_1}{1 - \omega^2 L_1 C'_T + j\omega R_1 C'_T}$$

where C'_T is the capacitance of C_1 in series with $(C_0 + C_L)$:

$$C'_T = \frac{C_1(C_0 + C_L)}{C_1 + C_0 + C_L}$$

The impedance of the crystal in *series* with the load capacitance is

$$Z_{XTAL + CL} = Z_{XTAL} + \frac{1}{j\omega C_L}$$

$$= \frac{C_L + C_1 + C_0}{j\omega C_L (C_1 + C_0)} \bullet \frac{1 - \omega^2 L_1 C'_T + j\omega R_1 C'_T}{1 - \omega^2 L_1 C_T + j\omega R_1 C_T}$$

where C_T and C'_T are as defined above.

The phase angles of these impedances are readily obtained from the impedance expressions themselves:

$$\theta_{XTAL} = \arctan \frac{\omega R_1 C_1}{1 - \omega^2 L_1 C_1}$$

$$- \arctan \frac{\omega R_1 C_T}{1 - \omega^2 L_1 C_T} - \frac{\pi}{2}$$

$$\theta_{XTAL\|C_L} = \arctan \frac{\omega R_1 C_1}{1 - \omega^2 L_1 C_1}$$

$$- \arctan \frac{\omega R_1 C'_T}{1 - \omega^2 L_1 C'_T} - \frac{\pi}{2}$$

$$\theta_{XTAL + C_L} = \arctan \frac{\omega R_1 C'_T}{1 - \omega^2 L_1 C'_T}$$

$$- \arctan \frac{\omega R_1 C_T}{1 - \omega^2 L_1 C_T} - \frac{\pi}{2}$$

The resonant ("series resonant") frequency is the frequency at which the phase angle is zero and the impedance is low. The antiresonant ("parallel resonant") frequency is the frequency at which the phase angle is zero and the impedance is high.

Each of the above θ-expressions contains two arctan functions. Setting the denominator of the argument of the first arctan function to zero gives (approximately) the "series resonant" frequency for that configuration. Setting the denominator of the argument of the second arctan function to zero gives (approximately) the "parallel resonant" frequency for that configuration.

For example, the resonant frequency of the crystal is the frequency at which

$$1 - \omega^2 L_1 C_1 = 0$$

Thus

$$\omega_s = \frac{1}{\sqrt{L_1 C_1}}$$

or

$$f_s = \frac{1}{2\pi\sqrt{L_1 C_1}}$$

It will be noted that the series resonant frequency of the "XTAL + CL" configuration (crystal in series with CL) is the same as the parallel resonant frequency of the "XTAL‖CL" configuration (crystal in parallel with C_L). This is the frequency at which

$$1 - \omega^2 L_1 C'_T = 0$$

Thus

$$\omega_a = \frac{1}{\sqrt{L_1 C'_T}}$$

or

$$f_a = \frac{1}{2\pi\sqrt{L_1 C'_T}}$$

This fact is used by crystal manufacturers in the process of calibrating a crystal to a specified load capacitance.

By subtracting the resonant frequency of the crystal from its antiresonant frequency, one can calculate the range of frequencies over which the crystal reactance is positive:

$$f_a - f_s = f_s(\sqrt{1 + C_1/C_0} - 1$$

$$f_s\left(\frac{C_1}{2C_0}\right)$$

Given typical values for C_1 and C_0, this range can hardly exceed 0.5% of fs. Unless the inverting amplifier in the positive reactance oscillator is doing something very strange indeed, the oscillation frequency is bound to be accurate to that percentage whether the crystal was calibrated for series operation or to any unspecified load capacitance.

Equivalent Series Resistance

ESR is the real part of Z_{XTAL} at the oscillation frequency. The oscillation frequency is the parallel resonant frequency of the "XTAL‖CL" configuration (which is the same as the series resonant frequency of the "XTAL + CL" configuration). Substituting this frequency into the Z_{XTAL} expression yields, after some algebraic manipulation,

$$ESR = \frac{R_1\left(\frac{C_0 + C_L}{C_L}\right)^2}{1 + \omega^2 C_1^2\left(\frac{C_0 + C_L}{C_L}\right)^2}$$

$$\cong R_1\left(1 + \frac{C_0}{C_L}\right)^2$$

Drive Level

The power dissipated by the crystal is $I_1^2 R_1$, where I_1 is the RMS current in the motional arm of the crystal. This current is given by $V_x/|Z_1|$, where V_x is the RMS voltage across the crystal, and $|Z_1|$ is the magnitude of the impedance of the motional arm. At the oscillation frequency, the motional arm is a positive (inductive) reactance in parallel resonance with $(C_0 + C_L)$. Therefore $|Z_1|$ is approximately equal to the magnitude of the reactance of $(C_0 + C_L)$:

$$|Z_1| = \frac{1}{2\pi f(C_0 + C_L)}$$

where f is the oscillation frequency. Then,

$$P = I_1^2 R_1 = \left(\frac{V_x}{|Z_1|}\right)^2 R_1$$

$$= [2\pi f (C_0 + C_L) V_x]^2 R_1$$

The waveform of the voltage across the crystal (XTAL1 to XTAL2) is approximately sinusoidal. If its peak value is VCC, then V_x is $VCC/\sqrt{2}$. Therefore,

$$P = 2R_1 [\pi f (C_0 + C_L) VCC]^2$$

APPENDIX B
OSCILLATOR ANALYSIS PROGRAM

The program is written in BASIC. BASIC is excruciatingly slow, but it has some advantages. For one thing, more people know BASIC than FORTRAN. In addition, a BASIC program is easy to develop, modify, and "fiddle around" with. Another important advantage is that a BASIC program can run on practically any small computer system.

Its slowness is a problem, however. For example, the routine which calculates the "start-up time constant" discussed in the text may take several hours to complete. A person who finds this program useful may prefer to convert it to FORTAN, if the facilities are available.

Limitations of the Program

The program was developed with specific reference to 8051-type oscillator circuitry. That means the on-chip amplifier is a simple inverter, and not a Schmitt Trigger. The 8096, the 80C51, the 80C48 and 80C49 all have simple inverters. The 8096 oscillator is almost identical to the 8051, differing mainly in the input protection circuitry. The CHMOS amplifiers have somewhat different parameters (higher gain, for example), and different transition levels than the 8051.

The MCS-48 family is specifically included in the program only to the extent that the input-output curve used in the steady-state analysis is that of a Schmitt Trigger, if the user identifies the device under analysis as an MCS-48 device. The analysis does not include the voltage dependent phase shift of the Schmitt Trigger.

The clamping action of the input protection circuitry is important in determining the steady-state amplitudes. The steady-state routine accounts for it by setting the negative peak of the XTAL1 signal at a level which depends on the amplitude of the XTAL1 signal in accordance with experimental observations. It's an exercise in curve-fitting. A user may find a different type of curve works better. Later steppings of the chips may behave differently in this respect, having somewhat different types of input protection circuitry.

It should be noted that the analysis ignores a number of important items, such as high-frequency effects in the on-chip circuitry. These effects are difficult to predict, and are no doubt dependent on temperature, frequency, and device sample. However, they can be simulated to a reasonable degree by adding an "output capacitance" of about 20 pF to the circuit model (i.e., in parallel with CX2) as described below.

Notes on Using the Program

The program asks the user to input values for various circuit parameters. First the crystal (or ceramic resonator) parameters are asked for. These are R1, L1, C1, and C0. The manufacturer can supply these values for selected samples. To obtain any kind of correlation between calculation and experiment, the values of these parameters must be known for the specific sample in the test circuit. The value that should be entered for C0 is the C0 of the crystal itself plus an estimated 7 pF to account for the XTAL1-to-XTAL2 pin capacitance, plus any other stray capacitance paralleling the crystal that the user may feel is significant enough to be included.

Then the program asks for the values of the XTAL1-to-ground and XTAL2-to-ground capacitances. For CXTAL1, enter the value of the externally connected bulk capacitor plus an estimated 7 pF for pin capacitance. For CXTAL2, enter the value of the externally connected bulk capacitor plus an estimated 7 pF for pin capacitance plus about 20 pF to simulate high-frequency roll-off and phase shifts in the on-chip circuitry.

Next the program asks for values for the small-signal parameters of the on-chip amplifier. Typically, for the 8051/8751,

Amplifier Gain Magnitude	= 15
Feedback Resistance	= 2300 KΩ
Output Resistance	= 2 KΩ

The same values can be used for MCS-48 (NMOS and HMOS) devices, but they are difficult to verify, because the Schmitt Trigger does not lend itself to small-signal measurements.

```
100 DEFDBL C,D,F,G,L,P,R,S,X
200 REM                                                          APRIL 8, 1983
300 REM ************************************************************************
400 REM
500 REM                        FUNCTIONS
600 REM
700 REM
800 REM FNZM(R,X) = MAGNITUDE OF A COMPLEX NUMBER, |R+jX|
900 DEF FNZM(R,X) = SQR(R^2+X^2)
1000 REM
1100 REM FNZP(R,X) = ANGLE OF A COMPLEX NUMBER
1200 REM           = 180/PI*ARCTAN(X/R)          IF R>0
1300 REM           = 180/PI*ARCTAN(X/R) + 180    IF R<0 AND X>0
1400 REM           = 180/PI*ARCTAN(X/R) - 180    IF R<0 AND X<0
1500 DEF FNZP(R,X) = 180/PI*ATN(X/R) - (SGN(R)-1)*SGN(X)*90
1600 REM
1700 REM    INDUCTIVE IMPEDANCE AT COMPLEX FREQUENCY S+jF (HZ)
1800 REM         Z = 2*PI*S*L  + j2*PI*F*L
1900 REM           = FNRL(S,L) + jFNXL(F,L)
2000 DEF FNRL(SL,LL) = 2##PI*SL*LL
2100 DEF FNXL(FL,LL) = 2##PI*FL*LL
2200 REM
2300 REM    CAPACITIVE IMPEDANCE AT COMPLEX FREQUENCY S+jF (HZ)
2400 REM         Z = 1/[2*PI*(S+jF)*C]
2500 REM           = S/[2*PI*(S^2+F^2)*C] + j(-F)/[2*PI*(S^2+F^2)C]
2600 REM           = FNRC(S,F,C) + jFNXC(S,F,C)
2700 DEF FNRC(SC,FC,CC) = SC/(2##PI*(SC^2+FC^2)*CC)
2800 DEF FNXC(SC,FC,CC) = -FC/(2##PI*(SC^2+FC^2)*CC)
2900 REM
3000 REM    RATIO OF TWO COMPLEX NUMBERS
3100 REM         RA+jXA      RA*RB+XA*XB       XA*RB-RA*XB
3200 REM         ------  =  -----------  + j  -----------
3300 REM         RB+jXB       RB^2+XB^2         RB^2+XB^2
3400 REM                  = FNRR(RA,XA,RB,XB) + jFNXR(RA,XA,RB,XB)
3500 DEF FNRR(RA,XA,RB,XB) = (RA*RB+XA*XB)/(RB^2+XB^2)
3600 DEF FNXR(RA,XA,RB,XB) = (XA*RB-XB*RA)/(RB^2+XB^2)
3700 REM
3800 REM    PRODUCT OF TWO COMPLEX NUMBERS
3900 REM        (RA+jXA)*(RB+jXB)  =  RA*RB-XA*XB + j(XA*RB+RA*XB)
4000 REM                           = FNRM(RA,XA,RB,XB) + jFNXM(RA,XA,RB,XB)
4100 DEF FNRM(RA,XA,RB,XB) = RA*RB - XA*XB
4200 DEF FNXM(RA,XA,RB,XB) = RA*XB + RB*XA
4300 REM
4400 REM
4500 REM    PARALLEL IMPEDANCES
4600 REM                                 (RA+jXA)*(RB+jXB)
4700 REM    (RA+jXA)||(RB+jXB)  =  -----------------
4800 REM                            RA+RB +j(XA+XB)
4900 REM
5000 REM      RA*(RB^2+XB^2)+RB*(RA^2+XA^2)        XA*(RB^2+XB^2)+XB*(RA^2+XA^2)
5100 REM   =  ------------------------------  +  j ------------------------------
5200 REM         (RA+RB)^2 + (XA+XB)^2                (RA+RB)^2 + (XA+XB)^2
5300 REM
5400 REM   = FNRP(RA,XA,RB,XB) + jFNXP(RA,XA,RB,XB)
5500 DEF FNRP(RA,XA,RB,XB) = (RA*(RB^2+XB^2) + RB*(RA^2+XA^2))/((RA+RB)^2 + (XA+XB)^2)
5600 DEF FNXP(RA,XA,RB,XB) = (XA*(RB^2+XB^2) + XB*(RA^2+XA^2))/((RA+RB)^2 + (XA+XB)^2)
5700 REM
5800 REM ************************************************************************
5900 REM
6000 REM                  BEGIN COMPUTATIONS
6100 REM
6200 LET PI = 3.141592654#
6300 REM
6400 REM              DEFINE CIRCUIT PARAMETERS
6500 GOSUB 14500
6600 REM
6700 REM  ESTABLISH NOMINAL RESONANT AND ANTIRESONANT CRYSTAL FREQUENCIES
6800 FS = FIX(1/(2*PI*SQR(L1*C1)))
6900 FA = FIX(1/(2*PI*SQR(L1*C1*CO/(C1+CO))))
7000 PRINT
7100 PRINT "XTAL IS SERIES RESONANT AT ";FS;" HZ"
7200 PRINT "      PARALLEL RESONANT AT ";FA;" HZ"
7300 PRINT
7400 PRINT "SELECT:  1  LIST PARAMETERS"
7500 PRINT "         2  CIRCUIT ANALYSIS"
7600 PRINT "         3  OSCILLATION FREQUENCY"
7700 PRINT "         4  START-UP TIME CONSTANT"
7800 PRINT "         5  STEADY-STATE ANALYSIS"
```

230659-44

```
7900 PRINT
8000 INPUT N
8100 IF N=1 THEN PRINT ELSE 8600
8200 REM
8300 REM ------------------- LIST PARAMETERS ----------------------------
8400 GOSUB 17100
8500 GOTO 6800
8600 IF N=2 THEN PRINT ELSE 9400
8700 REM
8800 REM --------------------- CIRCUIT ANALYSIS --------------------------
8900 PRINT " FREQUENCY S+JF  TYPE (S),(F) "
9000 INPUT SQ,FQ
9100 GOSUB 20200
9200 GOSUB 26600
9300 GOTO 6800
9400 IF N=3 THEN 10300 ELSE 11000
9500 REM
9600 REM ------------------ OSCILLATION FREQUENCY -------------------------
9700 CL = CX*CY/(CX+CY) + CO
9800 FQ = FIX(1/(2*PI*SQR(L1*C1*CL/(C1+CL))))
9900 SQ = 0
10000 DF = FIX(10^INT(LOG(FA-FS)/LOG(10)-2)+.5)
10100 DS = 0
10200 RETURN
10300 GOSUB 9700
10400 GOSUB 30300
10500 PRINT
10600 PRINT
10700 PRINT "FREQUENCY AT WHICH LOOP GAIN HAS ZERO PHASE ANGLE:"
10800 GOSUB 26600
10900 GOTO 6800
11000 IF N=4 THEN PRINT ELSE 12200
11100 REM
11200 REM ----------------- START-UP TIME CONSTANT -------------------------
11300 PRINT "THIS WILL TAKE SOME TIME ... ...."
11400 GOSUB 9700
11500 GOSUB 37700
11600 PRINT
11700 PRINT
11800 PRINT "FREQUENCY AT WHICH LOOP GAIN = 1 AT 0 DEGREES:"
11900 GOSUB 26600
12000 PRINT : PRINT "THIS YIELDS A START-UP TIME CONSTANT OF ";CSNG(1000000!/(2*PI*SQ));" MICROSECS"
12100 GOTO 6800
12200 IF N=5 THEN PRINT ELSE 7300
12300 REM
12400 REM ---------------- STEADY-STATE ANALYSIS --------------------------
12500 PRINT "STEADY-STATE ANALYSIS"
12600 PRINT
12700 PRINT "SELECT:   1. 8031/8051"
12800 PRINT "         2. 8751"
12900 PRINT "         3. 8035/8039/8040/8048/8049"
13000 PRINT "         4. 8748/8749"
13100 INPUT IC%
13200 IF IC%<1 OR IC%>4 THEN 12600
13300 GOSUB 46900
13400 GOTO 7300
13500 REM   SUBROUTINE BELOW DEFINES INPUT-OUTPUT CURVE OF OSCILLATOR CKT
13600 IF IC%>2 AND VO=5 AND VI<2 THEN RETURN
13700 VO = -10*VI + 15
13800 IF VO>5 THEN VO = 5
13900 IF VO< 2 THEN VO =  2
14000 IF IC%>2 AND VO>2 THEN VO = 5
14100 RETURN
14200 REM
14300 REM ***************************************************************
14400 REM
14500 REM              DEFINE CIRCUIT PARAMETERS
14600 REM
14700 INPUT " R1 (OHMS)";R1
14800 INPUT " L1 (HENRY)";L1
14900 INPUT " C1 (PF)";X
15000 C1 = X*1E-12
15100 INPUT " CO (PF)";X
15200 CO = X*1E-12
15300 INPUT " CXTAL1 (PF)";X
15400 CX = X*1E-12
15500 INPUT " CXTAL2 (PF)";X
15600 CY = X*1E-12
```

230659-45

```
15700 INPUT " GAIN FACTOR MAGNITUDE";AV#
15800 INPUT " AMP FEEDBACK RESISTANCE (K-OHMS)";X
15900 RX = X*1000#
16000 INPUT " AMP OUTPUT RESISTANCE (K-OHMS)";X
16100 RO = X*1000#
16200 REM
16300 REM
16400 REM          LIST CURRENT PARAMETER VALUES
16500 GOSUB 17100
16600 RETURN
16700 REM
16800 REM
16900 REM ************************************************************
17000 REM
17100 REM              LIST CURRENT PARAMETER VALUES
17200 REM
17300 PRINT
17400 PRINT "CURRENT PARAMETER VALUES: 1. R1 = ";R1," OHMS"
17500 PRINT "                          2. L1 = ";CSNG(L1)," HENRY"
17600 PRINT "                          3. C1 = ";CSNG(C1*1E+12)," PF"
17700 PRINT "                          4. CO = ";CSNG(CO*1E+12)," PF"
17800 PRINT "                          5. CXTAL1 = ";CSNG(CX*1E+12)," PF"
17900 PRINT "                          6. CXTAL2 = ";CSNG(CY*1E+12)," PF"
18000 PRINT "    7. AMPLIFIER GAIN MAGNITUDE = ";AV#
18100 PRINT "    8.       FEEDBACK RESISTANCE = ";CSNG(RX*.001)," K-OHMS"
18200 PRINT "    9.         OUTPUT RESISTANCE = ";CSNG(RO*.001)," K-OHMS"
18300 PRINT
18400 PRINT "TO CHANGE A PARAMETER VALUE, TYPE (PARAM NO.),(NEW VALUE)."
18500 PRINT "OTHERWISE, TYPE 0,0 "
18600 INPUT N%,X
18700 IF N%=0 THEN RETURN
18800 IF N%=1 THEN R1 = X
18900 IF N%=2 THEN L1 = X
19000 IF N%=3 THEN C1 = X*1E-12
19100 IF N%=4 THEN CO = X*1E-12
19200 IF N%=5 THEN CX = X*1E-12
19300 IF N%=6 THEN CY = X*1E-12
19400 IF N%=7 THEN AV# = X
19500 IF N%=8 THEN RX = X*1000!
19600 IF N%=9 THEN RO = X*1000!
19700 GOTO 17400
19800 REM
19900 REM
20000 REM ************************************************************
20100 REM
20200 REM                  CIRCUIT ANALYSIS
20300 REM
20400 REM  This routine calculates the loop gain at complex frequency SQ+jFQ.
20500 REM
20600 REM  1. Crystal impedance: RE + jXE
20700 REM
20800 X1 = FNXL(FQ,L1) + FNXC(SQ,FQ,C1)
20900 RE = FNRP((R1+FNRL(SQ,L1)+FNRC(SQ,FQ,C1)),X1,FNRC(SQ,FQ,CO),FNXC(SQ,FQ,CO))
21000 XE = FNXP((R1+FNRL(SQ,L1)+FNRC(SQ,FQ,C1)),X1,FNRC(SQ,FQ,CO),FNXC(SQ,FQ,CO))
21100 REM
21200 REM  2. RF + jXF  =  (RE+jXE)||(amplifier feedback resistance)
21300 REM
21400 RF = FNRP(RX,0,RE,XE)
21500 XF = FNXP(RX,0,RE,XE)
21600 REM
21700 REM  3. Input impedance: ZI = RI + jXI = impedance of CXTAL1
21800 REM
21900 RI = FNRC(SQ,FQ,CX)
22000 XI = FNXC(SQ,FQ,CX)
22100 REM
22200 REM  4. Load impedance: ZL = (impedance of CXTAL2)||[(RF+RI)+j(XF+XI)]
22300 REM
22400 RL = FNRP((RF+RI),(XF+XI),FNRC(SQ,FQ,CY),FNXC(SQ,FQ,CY))
22500 XL = FNXP((RF+RI),(XF+XI),FNRC(SQ,FQ,CY),FNXC(SQ,FQ,CY))
22600 REM
22700 REM  5. Amplifier gain  A = -AV*ZL/(ZL+RO)
22800 REM                       = A(real) + jA(imaginary)
22900 REM
23000 AR# = -AV#*FNRR(RL,XL,(RO+RL),XL)
23100 AI# = -AV#*FNXR(RL,XL,(RO+RL),XL)
23200 REM
23300 REM  6. Feedback ratio  (beta) = (RI+jXI)/[(RF+RI)+j(XF+XI)]
23400 REM                           = B(real) + jB(imaginary)
```

230659-46

```
23500 REM
23600 BR# = FNRR(RI,XI,(RI+RF),(XI+XF))
23700 BI# = FNXR(RI,XI,(RI+RF),(XI+XF))
23800 REM
23900 REM  7. Amplifier gain in magnitude/phase form:  AR+jAI = A at AP degrees
24000 REM
24100 A = FNZM(AR#,AI#)
24200 AP = FNZP(AR#,AI#)
24300 REM
24400 REM  8. (beta) in magnitude/phase form:  BR+jBI = B at BP degrees
24500 REM
24600 B = FNZM(BR#,BI#)
24700 BP = FNZP(BR#,BI#)
24800 REM
24900 REM  9. Loop gain:  G = (BR+jBI)*(AR+jAI)
25000 REM                  = G(real) + jG(imaginary)
25100 REM
25200 GR = FNRM(AR#,AI#,BR#,BI#)
25300 GI = FNXM(AR#,AI#,BR#,BI#)
25400 REM
25500 REM  10. Loop gain in magnitude/phase form:  GR+jGI = AL at AQ degrees
25600 REM
25700 AL = FNZM(GR,GI)
25800 AQ = FNZP(GR,GI)
25900 RETURN
26000 REM
26100 REM
26200 REM ************************************************************
26300 REM
26400 REM              PRINT CIRCUIT ANALYSIS RESULTS
26500 REM
26600 PRINT
26700 PRINT " FREQUENCY = ";SQ;" + J";FQ;" HZ"
26800 PRINT " XTAL IMPEDANCE = ";FNZM(RE,XE);" OHMS AT ";FNZP(RE,XE);" DEGREES"
26900 PRINT "         (RE = ";CSNG(RE);" OHMS)"
27000 PRINT "         (XE = ";CSNG(XE);" OHMS)"
27100 PRINT " LOAD IMPEDANCE = ";FNZM(RL,XL);" OHMS AT ";FNZP(RL,XL);" DEGREES"
27200 PRINT " AMPLIFIER GAIN = ";A;" AT ";AP;" DEGREES"
27300 PRINT " FEEDBACK RATIO = ";B;" AT ";BP;" DEGREES"
27400 PRINT " LOOP GAIN = ";AL;" AT ";AQ;" DEGREES"
27500 RETURN
27600 REM
27700 REM
27800 REM ************************************************************
27900 REM
28000 REM              SEARCH FOR FREQUENCY (S+JF)
28100 REM          AT WHICH LOOP GAIN HAS ZERO PHASE ANGLE
28200 REM
28300 REM  This routine searches for the frequency at which the imaginary part
28400 REM  of the loop gain is zero.  The algorithm is as follows:
28500 REM     1. Calculate the sign of the imaginary part of the loop gain (GI).
28600 REM     2. Increment the frequency.
28700 REM     3. Calculate the sign of GI at the incremented frequency.
28800 REM     4. If the sign of GI has not changed, go back to 2.
28900 REM     5. If the sign of GI has changed, and this frequency is within
29000 REM        1Hz of the previous sign-change, exit the routine.
29100 REM     6. Otherwise, divide the frequency increment by -10.
29200 REM     7. Go back to 2.
29300 REM  The routine is entered with the starting frequency SQ+jFQ and
29400 REM  starting increment DS+jDF already defined by the calling program.
29500 REM  In actual use either DS or DF is zero, so the routine searches for
29600 REM  a GI=0 point by incrementing either SQ or FQ while holding the other
29700 REM  constant.  It returns control to the calling program with the
29800 REM  incremented part of the frequency being within 1Hz of the actual
29900 REM  GI=0 point.
30000 REM
30100 REM  1. CALCULATE THE SIGN OF THE IMAGINARY PART OF THE LOOP GAIN (GI).
30200 REM
30300 GOSUB 20200
30400 GOSUB 26600
30500 IF GI=0 THEN RETURN
30600 SX% = INT(SGN(GI))
30700 IF SX%=+1 THEN DS = -DS
30800 REM  (REVERSAL OF DS FOR GI=0 IS FOR THE POLE-SEARCH ROUTINE.)
30900 REM
31000 REM  2. INCREMENT THE FREQUENCY.
31100 REM
31200 SP = SQ
```

230659-47

```
31300 FP = FQ
31400 SQ = SQ + DS
31500 FQ = FQ + DF
31600 REM
31700 REM  3  CALCULATE THE SIGN OF GI AT THE INCREMENTED FREQUENCY.
31800 REM
31900 GOSUB 20200
32000 GOSUB 26600
32100 IF INT(SGN(GI))=0 THEN RETURN
32200 REM
32300 REM  4  IF THE SIGN OF GI HAS NOT CHANGED, GO BACK TO 2.
32400 REM
32500 IF SX%+INT(SGN(GI))=0 THEN PRINT ELSE 31400
32600 SX% = -SX%
32700 REM
32800 REM  5  IF THE SIGN OF GI HAS CHANGED, AND IF THIS FREQUENCY IS WITHIN
32900 REM     1HZ OF THE PREVIOUS SIGN-CHANGE, AND IF GI IS NEGATIVE, THEN
33000 REM     EXIT THE ROUTINE  (THE ADDITIONAL REQUIREMENT FOR NEGATIVE GI
33100 REM     IS FOR THE POLE-SEARCH ROUTINE.)
33200 REM
33300 IF ABS(SP-SQ)<1 AND ABS(FP-FQ)<1 AND SX%=-1 THEN RETURN
33400 REM
33500 REM  6. DIVIDE THE FREQUENCY INCREMENT BY -10.
33600 REM
33700 DS = -DS/10#
33800 DF = -DF/10#
33900 REM
34000 REM  7. GO BACK TO 2.
34100 REM
34200 GOTO 31200
34300 REM
34400 REM
34500 REM ****************************************************************
34600 REM
34700 REM                SEARCH FOR POLE FREQUENCY
34800 REM
34900 REM  This routine searches for the frequency at which the loop gain = 1
35000 REM  at 0 degrees.  That frequency is the pole frequency of the closed-
35100 REM  loop gain function.  The pole frequency is a complex number, SQ+jFQ
35200 REM  (Hz).  Oscillator start-up ensues if SQ>0.  The algorithm is based on
35300 REM  the calculated behavior of the phase angle of the loop gain in the
35400 REM  region of interest on the complex plane.  The locus of points of zero
35500 REM  phase angle crosses the j-axis at the oscillation frequency and at
35600 REM  some higher frequency.  In between these two crossings of the j-axis,
35700 REM  the locus lies in Quadrant I of the complex plane, forming an
35800 REM  approximate parabola which opens to the left.  The basic plan is to
35900 REM  follow the locus from where it crosses the j-axis at the oscillation
36000 REM  frequency, into Quadrant I, and find the point on that locus where
36100 REM  the loop gain has a magnitude of 1.  The algorithm is as follows:
36200 REM     1. Find the oscillation frequency, 0+jFQ.
36300 REM     2. At this frequency calculate the sign of (AL-1).  (AL = magnitude
36400 REM        of loop gain.)
36500 REM     3. Increment FQ.
36600 REM     4. For this value of FQ, find the value of SQ for which the loop
36700 REM        gain has zero phase.
36800 REM     5. For this value of SQ+jFQ, calculate the sign of (AL-1).
36900 REM     6. If the sign of (AL-1) has not changed, go back to 3.
37000 REM     7. If the sign of (AL-1) has changed, and this value of FQ is
37100 REM        within 1Hz of the previous sign-change, exit the routine.
37200 REM     8. Otherwise, divide the FQ-increment by -10.
37300 REM     9. Go back to 3.
37400 REM
37500 REM  1. FIND THE OSCILLATION FREQUENCY, 0+jFQ.
37600 REM
37700 GOSUB 9700
37800 GOSUB 30300
37900 REM
38000 REM  2. AT THIS FREQUENCY, CALCULATE THE SIGN OF (AL-1).
38100 REM
38200 SY% = INT(SGN(AL-1!))
38300 IF SY%=-1 THEN STOP
38400 REM  ESTABLISH INITIAL INCREMENTATION VALUE FOR FQ.
38500 F1 = FQ
38600 DF = (FA-F1)/10#
38700 GOSUB 30300
38800 DE = (FQ-F1)/10#
38900 DF = 0
39000 FQ = F1
```

230659-48

```
39100 REM
39200 REM  3. INCREMENT FQ.
39300 REM
39400 FQ = FQ + DE
39500 REM
39600 REM  4. FOR THIS VALUE OF FQ, FIND THE VALUE OF SQ FOR WHICH THE LOOP
39700 REM     GAIN HAS ZERO PHASE.  (THE ROUTINE WHICH DOES THAT NEEDS DF = 0,
39800 REM     SO THAT IT CAN HOLD FQ CONSTANT, AND NEEDS AN INITIAL VALUE FOR
39900 REM     DS, WHICH IS ARBITRARILY SET TO DS = 1000.)
40000 REM
40100 DS = 1000#
40200 SQ = 0
40300 GOSUB 30300
40400 IF AL=1! THEN RETURN
40500 REM
40600 REM  5. FOR THIS VALUE OF SQ+jFQ, CALCULATE THE SIGN OF (AL-1).
40700 REM  6. IF THE SIGN OF (AL-1) HAS NOT CHANGED, GO BACK TO 3.
40800 REM
40900 IF SY%+INT(SGN(AL-1!))=0 THEN PRINT ELSE 39400
41000 REM
41100 REM  7. IF THE SIGN OF (AL-1) HAS CHANGED, AND THIS VALUE OF FQ IS WITHIN
41200 REM     1HZ OF THE PREVIOUS SIGN-CHANGE, EXIT THE ROUTINE.
41300 REM
41400 IF ABS(F1-FQ)<1 THEN RETURN
41500 REM
41600 REM  8. DIVIDE THE FQ-INCREMENT BY -10.
41700 REM
41800 DE = -DE/10#
41900 F1 = FQ
42000 SY% = -SY%
42100 REM
42200 REM  9. GO BACK TO 3.
42300 REM
42400 GOTO 39400
42500 REM
42600 REM
42700 REM ************************************************************
42800 REM
42900 REM                    STEADY-STATE ANALYSIS
43000 REM
43100 REM  The circuit model used in this analysis is similar to the one used
43200 REM  in the small-signal analysis, but differs from it in two respects.
43300 REM  First, it includes clamping and clipping effects described in the
43400 REM  text.  Second, the voltage source in the Thevenin equivalent of the
43500 REM  amplifier is controlled by the input voltage in accordance with an
43600 REM  input-output curve defined elsewhere in the program.
43700 REM    The analysis applies a sinusoidal input signal of arbitrary
43800 REM  amplitude, at the oscillation frequency, to the XTAL1 pin, then
43900 REM  calculates the resulting waveform from the voltage source.  Using
44000 REM  standard Fourier techniques, the fundamental frequency component of
44100 REM  this waveform is extracted.  This frequency component is then
44200 REM  multiplied by the factor |ZL/(ZL+RO)|, and the result is taken to be
44300 REM  the signal appearing at the XTAL2 pin.  This signal is then
44400 REM  multiplied by the feedback ratio (beta), and the result is taken to
44500 REM  be the signal appearing at the XTAL1 pin.  The algorithm is now
44600 REM  repeated using this computed XTAL1 signal as the assumed input
44700 REM  sinusoid.  Every time the algorithm is repeated, new values appear at
44800 REM  XTAL1 and XTAL2, but the values change less and less with each
44900 REM  repetition.  Eventually they stop changing.  This is the steady-state.
45000 REM    The algorithm is as follows:
45100 REM  1. Compute approximate oscillation frequency.
45200 REM  2. Call a circuit analysis at this frequency.
45300 REM  3. Find the quiescent levels at XTAL1 and XTAL2 (to establish the
45400 REM     beginning DC level at XTAL1).
45500 REM  4. Assume an initial amplitude for the XTAL1 signal.
45600 REM  5. Correct the DC level at XTAL1 for clamping effects, if necessary.
45700 REM  6. Using the appropriate input-output curve, extract a DC level and
45800 REM     the fundamental frequency component (multiplying the latter by
45900 REM     |ZL/(ZL+RO)|).
46000 REM  7. Clip off the negative portion of this output signal, if the
46100 REM     negative peak falls below zero.
46200 REM  8  If this signal, multiplied by (beta), differs from the input
46300 REM     amplitude by less than 1mV, or if the algorithm has been repeated
46400 REM     10 times, exit the routine
46500 REM  9. Otherwise, multiply the XTAL2 amplitude by (beta) and feed it
46600 REM     back to XTAL1, and go back to 5
46700 REM
46800 REM     1. COMPUTE APPROXIMATE OSCILLATION FREQUENCY.
```

230659-49

```
46900 GOSUB 9700
47000 REM
47100 REM      2. CALL A CIRCUIT ANALYSIS AT THIS FREQUENCY.
47200 GOSUB 20800
47300 PRINT : PRINT : PRINT "ASSUMED OSCILLATION FREQUENCY: "
47400 GOSUB 26600
47500 PRINT : PRINT
47600 REM
47700 REM      3. FIND QUIESCENT POINT
47800 REM   (At quiescence the voltages at XTAL1 and XTAL2 are equal. This
47900 REM    voltage level is found by trial-and-error, based on the input-
48000 REM    output curve, so that a person can change the input-output curve
48100 REM    as desired without having to re-calculate the quiescent point.)
48200 VI = 0
48300 VB = 1
48400 K1 = 1
48500 VI = VI + VB
48600 GOSUB 13600
48700 IF ABS(VO-VI)<.001 THEN 49200
48800 IF K1+SGN(VO-VI)=0 THEN 48900 ELSE 48500
48900 K1 = SGN(VO-VI)
49000 VB = -VB/10
49100 GOTO 48500
49200 VB = VI
49300 PRINT "QUIESCENT POINT = ";VB
49400 REM
49500 REM      4. ASSUME AN INITIAL AMPLITUDE FOR THE XTAL1 SIGNAL.
49600 EI = .01
49700 NR% = 0
49800 REM
49900 REM      5. CORRECT FOR CLAMPING EFFECTS, IF NECESSARY.
50000 REM   (K1 and K2 are curve-fitting parameters for the ROM parts.)
50100 K1 = (2.5-VB)/(3-VB)
50200 K2 = (VB-1.25)/(3-VB)
50300 IF IC%=2 OR IC%=4 THEN IF EI<(VB+.5) THEN EO = VB ELSE EO = EI - .5
50400 IF IC%=1 OR IC%=3 THEN IF EI<(VB+.5) THEN EO = VB ELSE EO = K1*EI+K2
50500 NR% = NR% + 1
50600 REM
50700 REM      6. DERIVE XTAL2 AMPLITUDE
50800 VO = 0
50900 VC = 0
51000 VS = 0
51100 FOR N% = -25 TO +24
51200 VI = EO - EI*COS(PI*N%/25)
51300 GOSUB 13600
51400 VO = VO + VO
51500 VC = VC + VO*COS(PI*N%/25)
51600 VS = VS + VO*SIN(PI*N%/25)
51700 NEXT N%
51800 VO = VO/50
51900 V1 = SQR(VC^2+VS^2)/25*FNZM(RL,XL)/FNZM((RL+RO),XL)
52000 REM
52100 REM      7. CLIP XTAL2 SIGNAL.
52200 IF VO-V1<0 THEN VL = 0 ELSE VL = VO-V1
52300 PRINT : PRINT "XTAL1 SWING = ";EO-EI;" TO ";EO+EI
52400 PRINT "XTAL2 SWING = ";VL;" TO ";VO+V1
52500 REM
52600 REM      8. TEST FOR TERMINATION.
52700 IF ABS(EI-V1*B)<.001 OR NR%=10 THEN RETURN
52800 REM
52900 REM      9. FEED BACK TO XTAL1 AND REPEAT
53000 EI = V1*B
53100 GOTO 50300
```

230659-50

INDEX

MCS®-48

8
8243 Expander, 2-4
8243 Port Characteristics, 2-5

A
Accumulator, 1-1
Addressing Beyond 2K, 2-1
Addressing External Data Memory, 2-4
ALE, 1-17, 2-9
ALU, 1-1
Arithmetic Logic Unit, 1-1

B
Bus, 1-5, 1-17, 2-9

C
Clock Circuits, 1-9
Conditional Branches, 1-6
Control Signals, 2-8
Counter, 1-7
Cycle Counter, 1-10

D
Data Memory, 1-3

E
EA, 1-15
Erasing EPROM, 1-18
Erasure Characteristics, 1-18
Event Counter, 1-9
Expansion of Data Memory, 2-3
Expansion of I/O, 2-4
Expansion of Program Memory, 2-1
Extended Addressing, 2-1
External Access Mode, 1-15
External Data Memory Addressing, 2-4
External Instruction Fetch, 2-1

I
I/O Expander Device (8243), 2-4
I/O Expansion, 2-4, 2-5
I/O Port Characteristics, 2-5
I/O Port Restore, 2-2
Instruction Decoder, 1-1
Instruction Fetch (External), 2-1
INT, 1-17
Interrupt, 1-5, 1-7
Interrupt Routines, 2-2
Interrupt Timing, 1-7

M
Memory Bank Switch, 2-1
Memory Bank Switching, 2-8
Memory Expansion, 2-5
Multi-Chip Systems, 2-7

O
Oscillator, 1-9

P
Pin Description, 1-16
Port 1, 1-5, 1-17
Port 2, 1-5, 2-9
Port Characteristics, 2-9
Power Down, 1-13
PROG, 1-17, 2-9
Program Counter, 1-5
Program Memory, 1-1
Program Status Word, 1-6
Programming EPROM, 1-18
PSEN, 2-9
PSW, 1-6

R
RD, 1-17, 2-9
Read Cycle, 2-3
Reset, 1-10, 1-17
Restoring I/O Ports, 2-2

S
Single Step, 1-11, 1-14
Stack, 1-5
State Counter, 1-10
Sync Mode, 1-15

T
T0, 1-5, 1-17
T1, 1-5, 1-17
Test Inputs, 1-5
Timer, 1-7, 1-9
Timing, 1-13
Timing Circuits, 1-9

V
VCC, 1-17
VDD, 1-17
Verifying EPROM, 1-18
VSS, 1-17

W
WR, 1-17, 2-9
Write Cycle, 2-3

MCS®-51

8
8031AH, 5-2
8032AH, 5-2
8051, 5-2
8051AH, 5-2
8052AH, 5-2
8052AH-BASIC, 5-2
80C31BH, 5-2
80C51BH, 5-2, 6-28
8751H, 5-2, 6-25, 6-26
8752BH, 5-2, 6-26
87C51, 5-2, 6-26, 6-27

A
AC (Auxiliary Carry) Flag, 6-3, 7-10
ACALL, 5-13, 7-24
Accumulator, 6-2
ADD, 7-25, 7-35
ADDC, 7-26, 7-35
ADDRESS/DATA Bus, 6-4, 6-5
AJMP, 5-13, 7-28
ALE, 5-16, 6-6, 6-30
ANL, 5-12, 6-6, 7-28
Arithmetic Instructions, 5-7
ASM51, 5-6

B
B Register, 6-2
Baud Rate, 6-12, 6-13, 6-14, 6-17
BCD, 5-8, 5-9, 5-10
Bit Addressable, 7-5
Boolean Instructions, 5-11, 5-12
Bus Cycle
  Data Memory, 5-16
  Program Memory, 5-16
Byte Addressable, 7-7

C
C (Carry) Flag, 5-6, 5-12, 6-3, 7-10
Capture Registers, 6-2
Case Jump, 5-7, 5-13
Ceramic Resonator, 5-14, 6-28
CJNE, 5-14, 7-31
CLR, 5-12, 6-6, 7-33
Control Registers, 6-2
CPL, 5-12, 6-6, 7-34
CPU Timing, 5-14
Crystal, 5-14, 6-27, 6-28

D
DA A, 5-8, 7-35
Data Memory, 5-3, 5-4, 5-10, 6-6, 6-30, 7-3
  Read Cycle, 6-31
  Write Cycle, 6-31
Data Pointer, 6-2
Data Transfers, 5-9
DEC, 6-6, 7-37
Direct Addressing, 5-6, 7-3, 7-5
DIV AB, 5-8, 6-22, 7-38
DJNZ, 5-14, 6-6, 7-39

E
EA (External Access), 5-4, 6-6, 6-21, 6-26, 6-30
Encryption Array, 6-26
EPROM Programming, 6-26, 6-30
EPROM Verifying, 6-26
Execution Times, 5-7
External Clock, 6-28, 6-29
External Program Execution, 5-4

F
Fetch/Execution Sequence, 5-15

I
I/O Buffers, 6-3, 6-4
Idle Mode, 5-2, 6-24, 6-25, 7-10
IE (Interrupt Enable), 5-17, 6-2, 6-20, 7-11
Immediate Constants, 5-7
INC, 6-6, 7-40
Indexed Addressing, 5-7
Indirect Addressing, 5-6, 7-3, 7-5
Instruction Set, 5-6, 7-20
Interrupt Response Time, 6-21, 6-22
Interrupts, 5-3, 5-17, 6-20, 7-11
  External, 6-22, 7-11
IP (Interrupt Priority), 5-17, 6-2, 6-21, 7-12

J
JB, 5-12, 7-42
JBC, 5-12, 6-6, 7-42
JC, 5-12, 7-43
JMP, 5-13, 7-44
JNB, 5-12, 7-45
JNC, 5-12, 7-45
JNZ, 7-46
Jump Instructions, 5-13
JZ, 7-46

L
LCALL, 5-13, 7-47
LJMP, 5-13, 7-47
Lock Bits, 6-26
Logical Instructions, 5-8
Lookup Tables, 5-7, 5-11

M
Machine Cycle, 5-15
MOV, 5-12, 6-6, 7-48
  16-bit, 5-9
MOVC, 5-11, 7-53
MOVX, 5-11, 5-16, 7-54
MUL AB, 5-8, 6-22, 7-56
Multiprocessor Communication, 6-11

N
Ninth Data Bit, 6-11, 6-17
NOP, 7-56

O
ONCE (On-circuit Emulation) Mode, 6-27
ORL, 5-12, 6-6, 7-57
Oscillator, 5-14, 6-23, 6-27
  External, 5-15
Oscillator Frequency, 6-12
OV (Overflow) Flag, 6-3, 7-10

P
P (Parity) Flag, 5-6, 6-3, 7-10
PCON, 6-2, 6-12, 6-24, 7-10, 7-19
Polling Sequence, 5-17
POP, 5-9, 7-60
Port Bit Latch, 6-3, 6-4
Ports, 6-2, 6-3, 6-29
Power Down Mode, 5-2, 6-24, 6-25, 7-10
PROG, 6-30
Program Memory, 5-2, 5-3, 6-6, 6-26, 6-30, 7-2
Program Memory Locks, 6-26
PSEN, 5-2, 5-16, 6-6, 6-30
PSW (Program Status Word), 5-6, 6-2, 7-10
PUSH, 5-9, 7-60

Q
Quick-Pulse Programming Algorithm, 6-26

R
RD signal, 5-3, 6-6
Register Banks, 5-6, 5-7, 6-3, 7-5, 7-10
Register Instructions, 5-7
Register-Specific Instructions, 5-7
Relative Offset, 5-12
Reset, 6-23, 6-30, 7-8
  Power-On, 6-24
RET, 5-13, 6-22, 7-61
RETI, 5-13, 6-21, 6-22, 7-61
RI (Receive Interrupt) Flag, 6-12, 6-14, 6-17, 6-20, 7-18
RL, 7-62
RLC, 7-62
RR, 7-63
RRC, 7-63

S
SBUF, 6-2, 6-10, 6-14
SCON, 6-2, 6-11, 6-14, 6-17, 7-18
Serial Port, 6-10, 6-15, 7-10, 7-12, 7-16, 7-18, 7-19
SET, 6-6
SETB, 5-12, 7-64
SFRs, 5-5, 6-1, 6-2, 6-4, 7-4, 7-5, 7-7, 7-8, 7-9, 7-10
SJMP, 5-13, 7-65
Stack Pointer, 6-2, 7-5
Start Bit, 6-11, 6-14, 6-17
State time, 5-15
Status Flags, 5-6
Stop Bit, 6-11, 6-14, 6-17
SUBB, 7-66
SWAP, 7-67
SWAP A, 5-9

T
T2CON, 6-2, 6-9, 6-10, 6-13, 7-16, 7-17
TCON, 6-2, 6-7, 6-8, 6-22, 7-11, 7-13
Third Priority Level, 5-18
TI (Transmit Interrupt) Flag, 6-12, 6-14, 6-17, 6-20, 7-18
Timer/Counters, 6-2, 6-6, 7-13
TMOD, 6-2, 6-7, 7-13, 7-14, 7-17

V
VPP, 6-26, 6-30

W
WR signal, 5-3, 6-6

X
XCH, 5-9, 5-10, 7-68
XCHD, 5-9, 5-10, 7-69
XRL, 5-12, 6-6, 7-69
XTAL1, 5-14, 5-15, 6-28, 6-29, 6-30
XTAL2, 5-14, 5-15, 6-28, 6-29, 6-30

MCS® -96

A
A-Bus, 12-2
A/D Converter, 12-32, 14-1, 14-8, 14-19
  Block Diagram, 14-9
  Commands, 12-33, 12-55
  Differences (8097 vs 8097BH), 14-15
  Input Circuit, 14-10
  Interface Suggestions, 14-9
  Reference Voltages, 14-11
  Results, 12-33, 12-55
  Transfer Function, 14-11
aa, 13-1
Absolute Error, 14-16
Actual Characteristic, 14-13, 14-15, 14-16
ADD, 13-2
ADDB, 13-3
ADDC, 13-4
ADDCB, 13-4
Address Decoding, 14-24
Address Valid, 12-12
Address/Command/Data Bus, 14-31
Addressing modes, 13-1
ADV, 12-12
ALE, 12-8, 14-8, 14-20, 14-26
ALU, 12-2
Analog Interface, 12-32
Analog Output, 14-17
Analog Reference Voltages, 14-11
Analog-to-Digital Conversion (A/D), 12-32, 14-9
AND, 13-5
ANDB, 13-6
ANGND, 14-1, 14-11
Auto Programming, 14-32

B
Back-bias Generator, 14-3
baop, 13-1
Baud Rates, 12-36, 12-55, 14-19
BEA, 13-1
BH, 13-1
BHE, 12-8, 14-8, 14-20, 14-24
Bit Operands, 12-14
bitno, 13-1
BR, 13-1
BR (Indirect), 13-7
Branch, 13-7
Break-Before-Make, 14-16
breg, 13-1
Bus (System), 12-8, 12-39, 14-8, 14-20
Bus Timings, 14-21
BUSWIDTH, 12-10, 14-24
Byte Operands, 12-14, 12-19

C
C Flag (see Carry Flag), 12-18
cadd, 13-1
CALL, 13-1, 13-26, 13-40
Carry Flag, 12-18, 13-8, 13-18, 13-22, 13-41
CCB (Chip Configuration Byte), 12-10, 14-36
CCR, 12-9, 12-57, 14-21, 14-36
CEA, 13-1
Channel-To-Channel Matching, 14-16
Characteristic, 14-16
Chip Configuration Byte (CCB), 12-10
Chip Configuration Register (see CCR), 12-9
Circuit
  A/D Input, 14-10
  Auto Programming, 14-32
  Configuration Byte Programming, 14-36
  D/A, 14-18
  Gang Programming, 14-33
  Oscillator, 14-2
  Reset, 14-4
Clear Instruction, 13-7, 13-8
Clearing the HSO, 12-31
CLKOUT, 12-3, 12-9, 14-8, 14-21
CLR, 13-7
CLRB, 13-8
CLRC, 13-8
CLRVT, 13-9
CMP, 12-19, 13-9
CMPB, 12-19, 13-10
Code, 14-16
Code Center, 14-16
Code Transition, 14-16
Code Width, 14-16
Compare Instruction, 13-9, 13-10
Complement Instruction, 13-35, 13-36
Conditional Jumps, 12-19
Configuration Byte Programming, 14-36
CPU, 12-1
Critical Regions, 12-26
Crosstalk, 14-16

D
D-Bus, 12-2
D.C. Input Leakage, 14-16
D/A Circuits, 14-18
D/A Converter, 14-17
Data Program Command, 14-34
Data Program/Verify Signals, 14-35
Data Verify Command, 14-35
DEC, 13-10
DECB, 13-11
Decrement Instruction, 13-10, 13-11
DI, 12-25, 12-27, 13-11
Differential Non-Linearity, 14-15, 14-16
Disabling The Watchdog, 12-41
DIV, 13-12
DIVB, 13-12
Divide, 13-12, 13-13
DIVU, 13-13
DIVUB, 13-13
DJNZ, 13-14
Double-Word Operands, 12-15, 12-19

E
EA Pin, 12-7, 12-10, 14-1, 14-30, 14-38
EI, 12-25, 12-27, 13-14
EPROM, 12-7
  Erasing, 14-39
  Lock, 12-13, 14-38
  Programming, 14-30
  Timings, 14-28
Erasing the 879XBH EPROM, 14-39
Examples
  Memory Systems, 14-25
  Port Reconstruction, 14-29
  Run-Time Programming, 14-37
  System Verification, 14-26
EXT, 12-19, 13-15
EXTB, 12-19, 13-15
External Clock Drive, 12-3, 14-2

F
Feedthrough, 14-15, 14-16
FIFO (see HSI), 12-29
Flag Settings, 13-1
Flags, 12-18
Full-Scale Error, 14-15, 14-16

G
Gang Programming
Auto, 14-32
Slave, 14-35

H
Hardware Connections, 14-1, 14-4
High Speed Inputs (see HSI), 12-28
High Speed Outputs (see HSO), 12-30
HSI, 12-28
Input Timings, 14-18
Interrupts, 12-29
Modes, 12-29, 12-55
Status, 12-29, 12-55
HSO, 12-34, 14-17
CAM, 12-30
Clearing, 12-31
Output Timings, 14-18
Pins, 14-8
Status, 12-30, 12-31

I
I Bit, 12-25
I/O Control Registers, 12-39
I/O Ports, 12-37
I/O Timings, 14-18
Ideal A/D Characteristic, 14-11, 14-16
Immediate Addressing, 12-16, 13-1
INC, 13-16
INCB, 13-16
Increment Instruction, 13-16
Indexed Addressing, 13-1
Indirect Addressing, 12-15, 13-1
Indirect Shifts, 13-1
Indirect with Auto-Increment, 12-16
Input Ports, 12-38, 14-7
Input Resistance, 14-16
INST Pin, 12-10, 14-20, 14-24
Instruction Set, 12-18, 13-1
Instruction Summary, 12-20, 12-50
Integer Operands, 12-14, 12-19
Interface Levels, 14-6
Internal ROM, 12-7
Internal Timings, 12-3, 14-3
Interrupt, 12-23, 12-24
Control, 12-25
Disable, 13-11
Enable, 13-14
Flags, 12-18
Global Disable, 12-25
HSI, 12-29
HSO, 12-31
Mask Register, 12-25
Nonmaskable, 12-4
Pending Register, 12-25, 12-27
Priorities, 12-25
Serial Port (TI/RI), 12-35, 14-19
Software Timers, 12-32
Timer, 12-28
Timing, 12-27
Vectors, 12-25, 12-56
IOC0, 12-39, 12-56
IOC1, 12-39, 12-56
IOS0, 12-39, 12-56
IOS1, 12-40, 12-56

J
JBC, 13-17
JBS, 13-18
JC, 13-18
JE, 13-19
JGE, 13-19
JGT, 12-19, 13-20
JH, 12-19, 13-1, 13-20
JLE, 13-21
JLT, 13-21
JNC, 13-22
JNE, 13-22
JNH, 13-23
JNST, 13-23
JNV, 13-24
JNVT, 13-24
JST, 13-25
Jump, 13-29, 13-50
Conditional, 12-19
on Bit Clear, 13-17
on Bit Set, 13-18
Jump to self, 14-37
JV, 13-25
JVT, 13-26

L
LCALL, 13-1, 13-26
LD, 13-27
LDB, 13-27
LDBSE, 12-19, 13-28
LDBZE, 12-19, 13-28
Least Significant Bit, 14-16
LJMP, 13-1, 13-29
Load, 13-27
Load Sign Extended, 13-28
Load Zero Extended, 13-28
Lock Modes, 12-13
Long-Indexed Addressing, 12-16
Long-Integer Operands, 12-15, 12-19
Ireg, 13-1
LSB - Least Significant Bit, 14-16

M
Memory
Controller, 12-8
Interface, 14-20
Map, 12-4, 12-49
Reserved Locations, 12-7
Timings, 12-9
Missed Codes, 14-15, 14-16
Modified Quick-Pulse Programming, 14-39
Monotonic, 14-15, 14-16
Monotonicity, 14-16
MUL, 13-29, 13-30
MULB, 13-30, 13-31
Multiply, 13-29, 13-30, 13-31, 13-32, 13-33
Multiprocessor Communications, 12-37
MULU, 13-31, 13-32
MULUB, 13-32, 13-33

N
N Flag (see Negative Flag), 12-18
NEG, 13-33
Negate Instruction, 13-33, 13-34
Negative Flag, 12-18
NEGB, 13-34
NMI, 12-4, 14-1
No Missed Codes, 14-15, 14-16
Noise Protection, 14-28
Non-Linearity, 14-15, 14-16
Nonmaskable Interrupt, 12-4
NOP, 13-34
NORML, 12-27
NORML (Normalize), 13-35
NOT, 13-35
NOTB, 13-36

O
Off-Isolation, 14-15, 14-17
Opcode, 13-1
Opcode List, 12-52
Open Drain Ports, 14-7
Operand Addressing, 12-15
Operand Types, 12-14
OR, 13-36
ORB, 13-37
Oscillator, 12-3, 14-1, 14-2
Output Ports, 12-39

Overflow Flag, 12-18, 13-24, 13-25
Overflow Trap, 12-18, 13-9, 13-24, 13-26

P
Packaging, 12-46, 12-47, 14-29
PACT (Programming Active), 14-31
PALE (Programming ALE Input), 14-31
PC (Program Counter), 12-2, 13-1
PCCB (Programming CCB), 12-10, 14-36
PDO (Program Duration Overflow), 14-31
Phase A, 12-3
Phases - Internal Clock, 12-3
Pin Description, 12-42
Pin List, 12-45
Pinouts, 12-48
PLMREG, 12-22
PMODE, 12-57, 14-30, 14-31
POP, 13-37
Pop Flags, 13-38
POPF, 12-27, 13-38
Port 0, 12-38
  Timings, 14-19
Port 1, 12-38
  Timings, 14-19
Port 2, 12-37, 12-38, 12-57
  Timings, 14-19
Port 3, 12-39, 14-31
  Timings, 14-19
Port 4, 12-39, 14-31
  Timings, 14-19
Port Reconstruction, 14-28
Power Down, 12-5, 14-6
PROG (Programming Pulse), 14-31
Program Counter (PC), 12-2, 13-1
Program Lock, 14-38
Program Status Word, 12-17
Program Verified, 14-31
Program/Verify Signals, 14-35
Programming, 14-30
  Auto, 14-32
  Configuration Byte, 14-36
  Gang-Auto, 14-32
  Gang-Slave, 14-35
  Mode Select, 14-31
  Modes, 14-30
  Run-Time, 14-37
  Slave, 14-34
Programming Active, 14-31
Programming Ale Input, 14-31
Programming Chip Configuration Byte, 12-10
Programming Duration Overflow, 14-31
Programming Pulse, 14-31
PSW, 12-17, 12-25
Pulse Width Modulation (see PWM), 12-33
PUSH, 13-38
Push Flags, 13-39
PUSHF, 12-27, 13-39
PVER (Program Verified Output), 14-31
PWM, 12-33, 14-8, 14-17

Q
Quasi-Bidirectional Port, 12-38, 14-6, 14-7
Queue, 12-8

R
RALU, 12-1, 12-2
RD, 12-9, 14-8
Read, 12-9, 14-8
Ready, 12-9, 12-13, 14-21
Receive Interrupt (RI), 12-35, 14-19
Register Direct Addressing, 12-15, 13-1
Register File, 12-1, 12-2, 12-4
REN (Receiver Enable), 12-35, 14-19
Repeatability, 14-17
Reserved Memory Locations, 12-7
Reset, 12-3, 12-5, 12-41, 14-3
  Instruction, 13-40, 14-28
  Multiple Chip, 14-4
  Sequence, 14-3
  Status, 12-42
  Sync Mode, 12-42
Resolution, 14-17
RET, 13-39
Return, 13-39
RI (Receive Interrupt), 12-35, 14-19
ROM/EPROM Dump Mode, 14-38
ROM/EPROM Lock, 12-13, 14-38
RST (Reset Instruction), 13-40, 14-28
Run-Time Programming, 14-37
RXD Pin, 14-8

S
SALE (Slave ALE), 14-31
Sample Delay, 14-17
Sample Delay Uncertainty, 14-17
Sample Time, 14-17
Sample Time Uncertainty, 14-17
Sample Window, 14-17
SCALL, 13-1, 13-40
Secuity Key, 14-38
Serial Port, 12-34
  Control/Status, 12-35
  Mode 0, 12-34
  Mode 0 Example, 14-20
  Mode 1, 12-35
  Mode 2, 12-35
  Mode 3, 12-35
  Timings, 14-19
SETC, 13-41
SFR, 12-1, 12-4, 12-5
SFR Summary, 12-55
Shift, 13-41
Shift Indirect, 13-1
SHL, 13-41
SHLB, 13-42
SHLL, 13-43
Short-Indexed Addressing, 12-16
Short-Integer Operands, 12-14, 12-19
SHR, 13-44
SHRA, 13-45
SHRAB, 13-46
SHRAL, 13-47
SHRB, 13-48
SHRL, 13-49
SID (Slave ID), 14-31
Sign Extend, 13-15
Signature Word, 14-39
SIGND, 12-27
SJMP, 13-1, 13-50
SKIP, 13-50
Slave Ale, 14-31
Slave ID Number, 14-31
Slave PC, 12-2, 12-8
Slave Programming, 14-34
Slave Programming Pulse, 14-31
Software, 12-14
Software Protection, 12-41
Software Standards, 12-22
Software Timers, 12-32
Software Trap, 13-55
Special Function Register (see SFR), 12-1
SPROG (Slave Programming Pulse), 14-31
SP_CON, 12-35, 12-55
SP_STAT, 12-35, 12-55
ST Flag (see Sticky Bit), 12-18
ST instruction, 13-51
Stack Pointer, 12-4, 12-17
Standard I/O Ports, 14-19
State time, 12-3
State Time List, 12-52
STB, 13-51
Sticky Bit, 12-18, 13-23, 13-25
Store, 13-51
SUB, 13-52
SUBB, 13-53

SUBC, 13-54
SUBCB, 13-54
Subroutine Linkage, 12-22
Subtract, 13-52, 13-53, 13-54
Successive Approximation, 14-17
System Bus, 12-8, 12-39, 14-8, 14-20
System Bus Timings, 14-21
System Verification, 14-26

T
T2CLK, 14-19
TAVDV, 14-27
TAVGY, 14-22
TAVLL, 14-23, 14-27
TAYDV, 14-22
TCHCH, 14-23
TCHCL, 14-23
TCLLH, 14-23
TCLYL, 14-23
TCLYX, 14-22
Temperature Coefficients, 14-17
Terminal Based Characteristic, 14-14, 14-17
Test, 14-1
THLHH, 14-23, 14-27
TI (Transmit Interrupt), 12-35, 14-19
Timer 1, 12-27
Timer 2, 12-27, 12-31
Timer Interrupts, 12-28
Timers, 12-27
Timings
  Definitions, 14-22, 14-23
  HSI/HSO, 14-18
  I/O, 14-18
  I/O Ports, 14-19
  Internal, 12-3, 14-3
  Serial Port, 14-19
  System Bus, 14-21
TLHLL, 14-23
TLLAX, 14-23
TLLCH, 14-23
TLLGV, 14-22
TLLGX, 14-22
TLLHL, 14-23
TLLRL, 14-23, 14-27
TLLYH, 14-22
TLLYL, 14-22
TOHCH, 14-23
TQVHL, 14-23
TQVWH, 14-23, 14-28
Transmit Interrupt (TI), 12-35, 14-19
TRAP, 12-27, 13-55
TRHBX, 14-23
TRHDZ, 14-22
TRHLH, 14-23
TRLDV, 14-22, 14-27
TRLRH, 14-23
TWHBX, 14-23
TWHLH, 14-23
TWHQX, 14-23
TWLWH, 14-23
TXD Pin, 14-8
TYLYH, 14-22

V
V Flag (see Overflow Flag), 12-18
VCC, 14-1
VCC Rejection, 14-15, 14-17
VPD, 12-5, 14-1
VREF, 14-1, 14-11
VSS, 14-1
VSS1, 14-1
VSS2, 14-1
VT Flag (see Overflow Trap), 12-18

W
waop, 13-1
Watchdog Timer, 12-40, 14-5
  Disabling, 12-41
Word Dump Command, 14-35
Word Operands, 12-14, 12-19
WR, 12-9, 14-8, 14-24
wreg, 13-1
WRH, 14-20
Write, 12-9
Write Strobe Mode, 12-12
WriteHigh, 14-24
WriteLow, 14-24
WRL, 14-20

X
XOR, 13-55
XORB, 13-56
XTAL Inputs, 12-3, 14-1, 14-2

Z
Z Flag (see Zero Flag), 12-18
Zero Flag, 12-18
Zero Offset, 14-15, 14-17
Zero Register, 12-17

LITERATURE

1987 HANDBOOKS (European Prices)

Product line handbooks contain data sheets, application notes, article reprints and other design information.

NAME	ORDER NUMBER	
MEMORY COMPONENTS HANDBOOK	210830	CATEGORY F
MICROCOMMUNICATIONS HANDBOOK	231658	CATEGORY F
EMBEDDED CONTROLLER HANDBOOK (includes Microcontrollers and 8085, 80186, 80188)	210918	CATEGORY F
MICROPROCESSOR AND PERIPHERAL HANDBOOK (2 Volume Set)	230843	CATEGORY H
DEVELOPMENT TOOLS HANDBOOK	210940	CATEGORY F
DOS DEVELOPMENT SOFTWARE CATALOG	280199	N/C
OEM BOARDS AND SYSTEMS HANDBOOK	280407	CATEGORY F
MILITARY HANDBOOK	210461	CATEGORY F
COMPONENTS QUALITY/RELIABILITY HANDBOOK	210997	CATEGORY F
SYSTEMS QUALITY/RELIABILITY HANDBOOK	231762	CATEGORY F
PRODUCT GUIDE Overview of Intel's complete product lines	210846	N/C
LITERATURE GUIDE List of Intel Literature	E00029	N/C
INTEL PACKAGING OUTLINES AND DIMENSIONS Packaging types, number of leads, etc.	231369	N/C

PRICE CODE	Finland FIM	Norway NKR	Sweden SEK	Denmark DKR	The Netherlands DFL	France FFR	U.K. PDS	Germany DM	Switzerland SFR	Austria AS
F	128.00	190.00	179.00	255.00	78.00	190.00	17.00	62.00	45.00	440.00
H	201.00	294.00	281.00	349.00	115.00	300.00	25.00	98.00	75.00	696.00

LOCAL TAX NOT INCLUDED

intel

EUROPEAN LITERATURE ORDER FORM

ORDER NUMBER	TITLE	QTY	PRICE	TOTAL
		X		
		X		
		X		
		X		
		X		
		X		
		X		
		X		
		X		
		X		
		X		

PAYMENT

Cheques should be made payable to your local Intel Sales Office (see inside back cover).

Other forms of payment may be available in your country. Please contact the Literature Co-ordinator at your local Intel Sales Office for details.

The Completed form should be marked for the attention of the LITERATURE CO-ORDINATOR and returned to your local Intel Sales Office.

SUB TOTAL _____

LOCAL TAX _____

TOTAL _____

NAME _____

COMPANY _____

ADDRESS _____

_____

_____

_____

PHONE NO. _____

DOMESTIC SALES OFFICES

ALABAMA

Intel Corp.
5015 Bradford Drive
Suite 2
Huntsville 35805
Tel: (205) 830-4010

ARIZONA

Intel Corp.
11225 N. 28th Drive
Suite 214D
Phoenix 85029
Tel: (602) 869-4980

Intel Corp.
1161 N. El Dorado Place
Suite 301
Tucson 85715
Tel: (602) 299-6815

CALIFORNIA

Intel Corp.
21515 Vanowen Street
Suite 116
Canoga Park 91303
Tel: (818) 704-8500

Intel Corp.
2250 E. Imperial Highway
Suite 218
El Segundo 90245
Tel: (213) 640-6040

Intel Corp.
1510 Arden Way, Suite 101
Sacramento 95815
Tel: (916) 920-8096

Intel Corp.
4350 Executive Drive
Suite 105
San Diego 92121
Tel: (619) 452-5880

Intel Corp.*
2000 East 4th Street
Suite 100
Santa Ana 92705
Tel: (714) 835-9642
TWX: 910-595-1114

Intel Corp.*
San Tomas 4
2700 San Tomas Expressway
Santa Clara, CA 95051
Tel: (408) 986-8086
TWX: 910-338-0255

COLORADO

Intel Corp.
3300 Mitchell Lane, Suite 210
Boulder 80301
Tel: (303) 442-8088

Intel Corp.
4445 Northpark Drive
Suite 100
Colorado Springs 80907
Tel: (303) 594-6622

Intel Corp.*
650 S. Cherry Street
Suite 915
Denver 80222
Tel: (303) 321-8086
TWX: 910-931-2289

CONNECTICUT

Intel Corp.
26 Mill Plain Road
Danbury 06810
Tel: (203) 748-3130
TWX: 710-456-1199

EMC Corp.
222 Summer Street
Stamford 06901
Tel: (203) 327-2934

FLORIDA

Intel Corp.
242 N. Westmonte Drive
Suite 105
Altamonte Springs 32714
Tel: (305) 869-5588

Intel Corp.
6363 N.W. 6th Way, Suite 100
Ft. Lauderdale 33309
Tel: (305) 771-0600
TWX: 510-956-9407

FLORIDA (Cont'd)

Intel Corp.
11300 4th Street North
Suite 170
St. Petersburg 33702
Tel: (813) 577-2413

GEORGIA

Intel Corp.
3280 Pointe Parkway
Suite 200
Norcross 30092
Tel: (404) 449-0541

ILLINOIS

Intel Corp.*
300 N. Martingale Road, Suite 400
Schaumburg 60172
Tel: (312) 310-8031

INDIANA

Intel Corp.
8777 Purdue Road
Suite 125
Indianapolis 46268
Tel: (317) 875-0623

IOWA

Intel Corp.
St. Andrews Building
1930 St. Andrews Drive N.E.
Cedar Rapids 52402
Tel: (319) 393-5510

KANSAS

Intel Corp.
8400 W. 110th Street
Suite 170
Overland Park 66210
Tel: (913) 345-2727

MARYLAND

Intel Corp.*
7321 Parkway Drive South
Suite C
Hanover 21076
Tel: (301) 796-7500
TWX: 710-862-1944

Intel Corp.
7833 Walker Drive
Greenbelt 20770
Tel: (301) 441-1020

MASSACHUSETTS

Intel Corp.*
Westford Corp. Center
3 Carlisle Road
Westford 01886
Tel: (617) 692-3222
TWX: 710-343-6333

MICHIGAN

Intel Corp.
7071 Orchard Lake Road
Suite 100
West Bloomfield 48033
Tel: (313) 851-8096

MINNESOTA

Intel Corp.
3500 W. 80th Street
Suite 360
Bloomington 55431
Tel: (612) 835-6722
TWX: 910-576-2867

MISSOURI

Intel Corp.
4203 Earth City Expressway
Suite 131
Earth City 63045
Tel: (314) 291-1990

NEW JERSEY

Intel Corp.*
Parkway 109 Office Center
328 Newman Springs Road
Red Bank 07701
Tel: (201) 747-2233

Intel Corp.
75 Livingston Avenue
First Floor
Roseland 07068
Tel: (201) 740-0111

NEW MEXICO

Intel Corp.
8500 Menual Boulevard N.E.
Suite B 295
Albuquerque 87112
Tel: (505) 292-8086

NEW YORK

Intel Corp.*
300 Vanderbilt Motor Parkway
Hauppauge 11788
Tel: (516) 231-3300
TWX: 510-227-6236

Intel Corp.
Suite 2B Hollowbrook Park
15 Myers Corners Road
Wappinger Falls 12590
Tel: (914) 297-6161
TWX: 510-248-0060

Intel Corp.*
850 Cross Keys Office Park
Fairport 14450
Tel: (716) 425-2750
TWX: 510-253-7391

NORTH CAROLINA

Intel Corp.
5700 Executive Center Drive
Suite 213
Charlotte 28212
Tel: (704) 568-8966

Intel Corp.
2700 Wycliff Road
Suite 102
Raleigh 27607
Tel: (919) 781-8022

OHIO

Intel Corp.*
3401 Park Center Drive
Suite 220
Dayton 45414
Tel: (513) 890-5350
TWX: 810-450-2528

Intel Corp.*
25700 Science Park Drive
Beachwood 44122
Tel: (216) 464-2736
TWX: 810-427-9298

OKLAHOMA

Intel Corp.
6801 N. Broadway
Suite 115
Oklahoma City 73116
Tel: (405) 848-8086

OREGON

Intel Corp.
15254 N.W. Greenbrier Parkway, Bldg. B
Beaverton 97006
Tel: (503) 645-8051
TWX: 910-467-8741

PENNSYLVANIA

Intel Corp.
1513 Cedar Cliff Drive
Camphill 17011
Tel: (717) 737-5035

Intel Corp.*
455 Pennsylvania Avenue
Fort Washington 19034
Tel: (215) 641-1000
TWX: 510-661-2077

Intel Corp.*
400 Penn Center Boulevard
Suite 610
Pittsburgh 15235
Tel: (412) 823-4970

PUERTO RICO

Intel Microprocessor Corp.
South Industrial Park
P.O. Box 910
Las Piedras 00671
Tel: (809) 733-3030

TEXAS

Intel Corp.
313 E. Anderson Lane
Suite 314
Austin 78752
Tel: (512) 454-3628

Intel Corp.*
12300 Ford Road
Suite 380
Dallas 75234
Tel: (214) 241-8087
TWX: 910-860-5617

Intel Corp.*
7322 S.W. Freeway
Suite 1490
Houston 77074
Tel: (713) 988-8086
TWX: 910-881-2490

Industrial Digital Systems Corp.
5925 Sovereign
Suite 101
Houston 77036
Tel: (713) 988-9421

UTAH

Intel Corp.
5201 Green Street
Suite 290
Murray 84123
Tel: (801) 263-8051

VIRGINIA

Intel Corp.
1603 Santa Rosa Road
Suite 109
Richmond 23288
Tel: (804) 282-5668

WASHINGTON

Intel Corp.
155-108 Avenue N.E.
Suite 386
Bellevue 98004
Tel: (206) 453-8086
TWX: 910-443-3002

Intel Corp.
408 N. Mullan Road
Suite 102
Spokane 99206
Tel: (509) 928-8086

WISCONSIN

Intel Corp.
450 N. Sunnyslope Road
Suite 130
Chancellory Park 1
Brookfield 53005
Tel: (414) 784-8087

CANADA

BRITISH COLUMBIA

Intel Semiconductor of Canada, Ltd.
301-2245 W. Broadway
Vancouver V6K 2E4
Tel: (604) 738-6522

ONTARIO

Intel Semiconductor of Canada, Ltd.
2650 Queensview Drive
Suite 250
Ottawa K2B 8H6
Tel: (613) 829-9714
TELEX: 053-4115

Intel Semiconductor of Canada, Ltd.
190 Attwell Drive
Suite 500
Rexdale M9W 6H8
Tel: (416) 675-2105
TELEX: 06983574

QUEBEC

Intel Semiconductor of Canada, Ltd.
620 St. Jean Boulevard
Pointe Claire H9R 3K3
Tel: (514) 694-9130
TWX: 514-694-9134

*Field Application Location

CG-11/6/86

DOMESTIC DISTRIBUTORS

ALABAMA

Arrow Electronics, Inc.
1015 Henderson Road
Huntsville 35805
Tel: (205) 837-6955

†Hamilton/Avnet Electronics
4812 Commercial Drive N.W.
Huntsville 35805
Tel: (205) 837-7210
TWX: 810-726-2162

Pioneer/Technologies Group Inc.
4825 University Square
Huntsville 35805
Tel: (205) 837-9300
TWX: 810-726-2197

ARIZONA

†Hamilton/Avnet Electronics
505 S. Madison Drive
Tempe 85281
Tel: (602) 231-5100
TWX: 910-950-0077

Kierulff Electronics
4134 E. Wood Street
Phoenix 85040
Tel: (602) 437-0750
TWX: 910-951-1550

Wyle Distribution Group
17855 N. Black Canyon Highway
Phoenix 85023
Tel: (602) 866-2888

CALIFORNIA

Arrow Electronics, Inc.
19748 Dearborn Street
Chatsworth 91311
Tel: (818) 701-7500
TWX: 910-493-2086

Arrow Electronics, Inc.
1502 Crocker Avenue
Hayward 94544
Tel: (408) 487-4600

Arrow Electronics
9511 Ridgehaven Court
San Diego 92123
Tel: (619) 565-4800
TLX: 888064

†Arrow Electronics, Inc.
521 Weddell Drive
Sunnyvale 94086
Tel: (408) 745-6600
TWX: 910-339-9371

Arrow Electronics, Inc.
2961 Dow Avenue
Tustin 92680
Tel: (714) 838-5422
TWX: 910-595-2860

†Avnet Electronics
350 McCormick Avenue
Costa Mesa 92626
Tel: (714) 754-6051
TWX: 910-595-1928

Hamilton/Avnet Electronics
1175 Bordeaux Drive
Sunnyvale 94086
Tel: (408) 743-3300
TWX: 910-339-9332

†Hamilton/Avnet Electronics
4545 Viewridge Avenue
San Diego 92123
Tel: (619) 571-7500
TWX: 910-595-2638

†Hamilton/Avnet Electronics
20501 Plummer Street
Chatsworth 91311
Tel: (818) 700-6271
TWX: 910-494-2207

†Hamilton/Avnet Electronics
4103 Northgate Boulevard
Sacramento 95834
Tel: (916) 920-3150

Hamilton/Avnet Electronics
3002 G Street
Ontario 91311
Tel: (714) 989-9411

Hamilton/Avnet Electronics
19515 So. Vermont Avenue
Torrance 90502
Tel: (213) 515-3909
TWX: 910-349-6263

Hamilton Electro Sales
9650 De Soto Avenue
Chatsworth 91311
Tel: (818) 700-6500

†Hamilton Electro Sales
10950 W. Washington Boulevard
Culver City 90230
Tel: (213) 558-2458
TWX: 910-340-6364

CALIFORNIA (Cont'd)

Hamilton Electro Sales
1361 B West 190th Street
Gardena 90248
Tel: (213) 558-2131

†Hamilton Electro Sales
3170 Pullman Street
Costa Mesa 92626
Tel: (714) 641-4150
TWX: 910-595-2638

Kierulff Electronics
10824 Hope Street
Cypress 90430
Tel: (714) 220-6300

Kierulff Electronics, Inc.
1180 Murphy Avenue
San Jose 95131
Tel: (408) 971-2600
TWX: 910-379-6430

Kierulff Electronics, Inc.
14101 Franklin Avenue
Tustin 92680
Tel: (714) 731-5711
TWX: 910-595-2599

†Kierulff Electronics, Inc.
5650 Jillson Street
Commerce 90040
Tel: (213) 725-0325
TWX: 910-580-3666

Wyle Distribution Group
26560 Agoura Street
Calabasas 91302
Tel: (818) 880-9000
TWX: 818-372-0232

†Wyle Distribution Group
124 Maryland Street
El Segundo 90245
Tel: (213) 322-8100
TWX: 910-348-7140 or 7111

†Wyle Distribution Group
17872 Cowan Avenue
Irvine 92714
Tel: (714) 843-9953
TWX: 910-595-1572

†Wyle Distribution Group
11151 Sun Center Drive
Rancho Cordova 95670
Tel: (916) 638-5282

†Wyle Distribution Group
9525 Chesapeake Drive
San Diego 92123
Tel: (619) 565-9171
TWX: 910-335-1590

†Wyle Distribution Group
3000 Bowers Avenue
Santa Clara 95051
Tel: (408) 727-2500
TWX: 910-338-0296

Wyle Military
18910 Teller Avenue
Irvine 92750
Tel: (714) 851-9958
TWX: 310-371-9127

Wyle Systems
7382 Lampson Avenue
Garden Grove 92641
Tel: (714) 851-9953
TWX: 910-595-2642

COLORADO

Arrow Electronics, Inc.
1390 S. Potomac Street
Suite 136
Aurora 80012
Tel: (303) 696-1111

†Hamilton/Avnet Electronics
8765 E. Orchard Road
Suite 708
Englewood 80111
Tel: (303) 740-1017
TWX: 910-935-0787

†Wyle Distribution Group
451 E. 124th Avenue
Thornton 80241
Tel: (303) 457-9953
TWX: 910-936-0770

CONNECTICUT

†Arrow Electronics, Inc.
12 Beaumont Road
Wallingford 06492
Tel: (203) 265-7741
TWX: 710-476-0162

†Hamilton/Avnet Electronics
Commerce Industrial Park
Commerce Drive
Danbury 06810
Tel: (203) 797-2800
TWX: 710-456-9974

CONNECTICUT (Cont'd)

†Pioneer Northeast Electronics
112 Main Street
Norwalk 06851
Tel: (203) 853-1515
TWX: 710-468-3373

FLORIDA

†Arrow Electronics, Inc.
350 Fairway Drive
Deerfield Beach 33441
Tel: (305) 429-8200
TWX: 510-955-9456

†Arrow Electronics, Inc.
1001 N.W. 62nd Street
Suite 108
Ft. Lauderdale 33309
Tel: (305) 776-7790
TWX: 510-955-9456

†Arrow Electronics, Inc.
50 Woodlake Drive W., Bldg. B
Palm Bay 32905
Tel: (305) 725-1480
TWX: 510-959-6337

†Hamilton/Avnet Electronics
6801 N.W. 15th Way
Ft. Lauderdale 33309
Tel: (305) 971-2900
TWX: 510-956-3097

†Hamilton/Avnet Electronics
3197 Tech Drive North
St. Petersburg 33702
Tel: (813) 576-3930
TWX: 810-863-0374

Hamilton/Avnet Electronics
6947 University Boulevard
Winterpark 32792
Tel: (305) 628-3888
TWX: 810-853-0322

†Pioneer Electronics
221 N. Lake Boulevard
Suite 412
Alta Monte Springs 32701
Tel: (305) 834-9090
TWX: 810-853-0284

†Pioneer Electronics
674 S. Military Trail
Deerfield Beach 33442
Tel: (305) 428-8877
TWX: 510-955-9653

GEORGIA

†Arrow Electronics, Inc.
3155 Northwoods Parkway, Suite A
Norcross 30071
Tel: (404) 449-8252
TWX: 810-766-0439

Hamilton/Avnet Electronics
5825 D. Peachtree Corners
Norcross 30092
Tel: (404) 447-7500
TWX: 810-766-0432

Pioneer Electronics
5835B Peachtree Corners E
Norcross 30092
Tel: (404) 448-1711
TWX: 810-766-4515

ILLINOIS

†Arrow Electronics, Inc.
2000 E. Alonquin Street
Schaumberg 60195
Tel: (312) 397-3440
TWX: 910-291-3544

†Hamilton/Avnet Electronics
1130 Thorndale Avenue
Bensenville 60106
Tel: (312) 860-7780
TWX: 910-227-0060

MTI Systems Sales
1100 West Thorndale
Itasca 60143
Tel: (312) 773-2300

†Pioneer Electronics
1551 Carmen Drive
Elk Grove Village 60007
Tel: (312) 437-9680
TWX: 910-222-1834

INDIANA

†Arrow Electronics, Inc.
2495 Directors Row, Suite H
Indianapolis 46241
Tel: (317) 243-9353
TWX: 810-341-3119

Hamilton/Avnet Electronics
485 Gradle Drive
Carmel 46032
Tel: (317) 844-9333
TWX: 810-260-3966

INDIANA (Cont'd)

†Pioneer Electronics
6408 Castleplace Drive
Indianapolis 46250
Tel: (317) 849-7300
TWX: 810-260-1794

KANSAS

†Hamilton/Avnet Electronics
9219 Quivera Road
Overland Park 66215
Tel: (913) 888-8900
TWX: 910-743-0005

KENTUCKY

Hamilton/Avnet Electronics
1051 D. Newton Park
Lexington 40511

MARYLAND

Arrow Electronics, Inc.
8300 Gulford Road #H
Rivers Center
Columbia 21046
Tel: (301) 995-0003
TWX: 710-236-9005

†Hamilton/Avnet Electronics
6822 Oak Hall Lane
Columbia 21045
Tel: (301) 995-3500
TWX: 710-862-1861

†Mesa Technology Corporation
16021 Industrial Drive
Gaithersburg 20877
Tel: (301) 948-4350
Twx: 710-828-9702

†Pioneer Electronics
9100 Gaither Road
Gaithersburg 20877
Tel: (301) 948-0710
TWX: 710-828-0545

MASSACHUSETTS

†Arrow Electronics, Inc.
1 Arrow Drive
Woburn 01801
Tel: (617) 933-8130
TWX: 710-393-6770

†Hamilton/Avnet Electronics
10D Centennial Drive
Peabody 01960
Tel: (617) 532-3701
TWX: 710-393-0382

MTI Systems Sales
13 Fortune Drive
Billerica 01821

Pioneer Northeast Electronics
44 Hartwell Avenue
Lexington 02173
Tel: (617) 863-1200
TWX: 710-326-6617

MICHIGAN

Arrow Electronics, Inc.
755 Phoenix Drive
Ann Arbor 48104
Tel: (313) 971-8220
TWX: 810-223-6020

†Hamilton/Avnet Electronics
32487 Schoolcraft Road
Livonia 48150
Tel: (313) 522-4700
TWX: 810-242-8775

Hamilton/Avnet Electronics
2215 29th Street S.E.
Space A5
Grand Rapids 49508
Tel: (616) 243-8805
TWX: 810-273-6921

†Pioneer Electronics
13485 Stamford
Livonia 48150
Tel: (313) 525-1800
TWX: 810-242-3271

MINNESOTA

†Arrow Electronics, Inc.
5230 W. 73rd Street
Edina 55435
Tel: (612) 830-1800
TWX: 910-576-3125

Hamilton/Avnet Electronics
10300 Bren Road East
Minnetonka 55343
Tel: (612) 932-0600
TWX: (910) 576-2720

†Pioneer Electronics
10203 Bren Road East
Minnetonka 55343
Tel: (612) 935-5444
TWX: 910-576-2738

†Microcomputer System Technical Demonstrator Centers

DOMESTIC DISTRIBUTORS

MISSOURI

†Arrow Electronics, Inc.
2380 Schuetz
St. Louis 63141
Tel: (314) 567-6888
TWX: 910-764-0882

†Hamilton/Avnet Electronics
13743 Shoreline Court
Earth City 63045
Tel: (314) 344-1200
TWX: 910-762-0684

NEW HAMPSHIRE

†Arrow Electronics, Inc.
3 Perimeter Road
Manchester 03103
Tel: (603) 668-6968
TWX: 710-220-1684

Hamilton/Avnet Electronics
444 E. Industrial Drive
Manchester 03104
Tel: (603) 624-9400

NEW JERSEY

†Arrow Electronics, Inc.
6000 Lincoln East
Marlton 08053
Tel: (609) 596-8000
TWX: 710-897-0829

†Arrow Electronics, Inc.
2 Industrial Road
Fairfield 07006
Tel: (201) 575-5300
TWX: 710-998-2206

†Hamilton/Avnet Electronics
1 Keystone Avenue
Bldg. 36
Cherry Hill 08003
Tel: (609) 424-0110
TWX: 710-940-0262

†Hamilton/Avnet Electronics
10 Industrial
Fairfield 07006
Tel: (201) 575-3390
TWX: 701-734-4388

†Pioneer Northeast Electronics
45 Route 46
Pinebrook 07058
Tel: (201) 575-3510
TWX: 710-734-4382

†MTI Systems Sales
383 Route 46 W
Fairfield 07006
Tel: (201) 227-5552

NEW MEXICO

Alliance Electronics Inc.
11030 Cochiti S.E.
Albuquerque 87123
Tel: (505) 292-3360
TWX: 910-989-1151

Hamilton/Avnet Electronics
2524 Baylor Drive S.E.
Albuquerque 87106
Tel: (505) 765-1500
TWX: 910-989-0614

NEW YORK

†Arrow Electronics, Inc.
25 Hub Drive
Melville 11747
Tel: (516) 694-6800
TWX: 510-224-6126

†Arrow Electronics, Inc.
3375 Brighton-Henrietta Townline Road
Rochester 14623
Tel: (716) 427-0300
TWX: 510-253-4766

Arrow Electronics, Inc.
7705 Maltage Drive
Liverpool 13088
Tel: (315) 652-1000
TWX: 710-545-0230

Arrow Electronics, Inc.
20 Oser Avenue
Hauppauge 11788
Tel: (516) 231-1000
TWX: 510-227-6623

Hamilton/Avnet Electronics
333 Metro Park
Rochester 14623
Tel: (716) 475-9130
TWX: 510-253-5470

Hamilton/Avnet Electronics
103 Twin Oaks Drive
Syracuse 13206
Tel: (315) 437-2641
TWX: 710-541-1560

NEW YORK (Cont'd)

†Hamilton/Avnet Electronics
933 Motor Parkway
Hauppauge 11788
Tel: (516) 231-9800
TWX: 510-224-6166

†MTI Systems Sales
38 Harbor Park Drive
P.O. Box 271
Port Washington 11050
Tel: (516) 621-6200
TWX: 510-223-0846

†Pioneer Northeast Electronics
1806 Vestal Parkway East
Vestal 13850
Tel: (607) 748-8211
TWX: 510-252-0893

†Pioneer Northeast Electronics
60 Crossway Park West
Woodbury, Long Island 11797
Tel: (516) 921-8700
TWX: 510-221-2184

Pioneer Northeast Electronics
840 Fairport Park
Fairport 14450
Tel: (716) 381-7070
TWX: 510-253-7001

NORTH CAROLINA

Arrow Electronics, Inc.
5240 Greendairy Road
Raleigh 27604
Tel: (919) 876-3132
TWX: 510-928-1856

†Hamilton/Avnet Electronics
3510 Spring Forest Drive
Raleigh 27604
Tel: (919) 878-0819
TWX: 510-928-1836

Pioneer Electronics
9801 A-Southern Pine Boulevard
Charlotte 28210
Tel: (704) 524-8188
TWX: 810-621-0366

OHIO

Arrow Electronics, Inc.
7620 McEwen Road
Centerville 45459
Tel: (513) 435-5563
TWX: 810-459-1611

†Arrow Electronics, Inc.
6238 Cochran Road
Solon 44139
Tel (216) 248-3990
TWX: 810-427-9409

†Hamilton/Avnet Electronics
954 Senate Drive
Dayton 45459
Tel: (513) 433-0610
TWX: 810-450-2531

†Hamilton/Avnet Electronics
4588 Emery Industrial Parkway
Warrensville Heights 44128
Tel: (216) 831-3500
TWX: 810-427-9452

†Pioneer Electronics
4433 Interpoint Boulevard
Dayton 45424
Tel: (513) 236-9900
TWX: 810-459-1622

†Pioneer Electronics
4800 E. 131st Street
Cleveland 44105
Tel: (216) 587-3600
TWX: 810-422-2211

OKLAHOMA

Arrow Electronics, Inc.
4719 S. Memorial Drive
Tulsa 74145
Tel: (918) 665-7700

OREGON

†Almac Electronics Corporation
1885 N.W. 169th Place
Beaverton 97006
Tel: (503) 629-8090
TWX: 910-467-8746

Hamilton/Avnet Electronics
6024 S.W. Jean Road
Bldg. C, Suite 10
Lake Oswego 97034
Tel: (503) 635-7848
TWX: 910-455-8179

Wyle Distribution Group
5250 N.E. Elam Young Parkway
Suite 600
Hillsboro 97124
Tel: (503) 640-6000
TWX: 910-460-2203

PENNSYLVANIA

Arrow Electronics, Inc.
650 Seco Road
Monroeville 15146
Tel: (412) 856-7000

Pioneer Electronics
259 Kappa Drive
Pittsburgh 15238
Tel: (412) 782-2300
TWX: 710-795-3122

†Pioneer Electronics
261 Gibralter Road
Horsham 19044
Tel: (215) 674-4000
TWX: 510-665-6778

TEXAS

†Arrow Electronics, Inc.
3220 Commander Drive
Carrollton 75006
Tel: (214) 380-6464
TWX: 910-860-5377

†Arrow Electronics, Inc.
10899 Kinghurst
Suite 100
Houston 77099
Tel: (713) 530-4700
TWX: 910-880-4439

Arrow Electronics, Inc.
10125 Metropolitan
Austin 78758
Tel: (512) 835-4180
TWX: 910-874-1348

†Hamilton/Avnet Electronics
1807 W. Braker Lane
Austin 78758
Tel: (512) 837-8911
TWX: 910-874-1319

†Hamilton/Avnet Electronics
2111 W. Walnut Hill Lane
Irving 75062
Tel: (214) 659-4100
TWX: 910-860-5929

†Hamilton/Avnet Electronics
4850 Wright Road #190
Houston 77477
Tel: (713) 780-1771
TWX: 910-881-5523

†Pioneer Electronics
9901 Burnet Road
Austin 78758
Tel: (512) 835-4000
TWX: 910-874-1323

Pioneer Electronics
13710 Omega Road
Dallas 75234
Tel: (214) 386-7300
TWX: 910-850-5563

Pioneer Electronics
5853 Point West Drive
Houston 77036
Tel: (713) 988-5555
TWX: 910-881-1606

UTAH

†Hamilton/Avnet Electronics
1585 West 2100 South
Salt Lake City 84119
Tel: (801) 972-2800
TWX: 910-925-4018

Wyle Distribution Group
1959 South 4130 West, Unit B
Salt Lake City 84104
Tel: (801) 974-9953

WASHINGTON

†Almac Electronics Corporation
14360 S.E. Eastgate Way
Bellevue 98007
Tel: (206) 643-9992
TWX: 910-444-2067

Arrow Electronics, Inc.
14320 N.E. 21st Street
Bellevue 98007
Tel: (206) 643-4800
TWX: 910-444-2017

Hamilton/Avnet Electronics
14212 N.E. 21st Street
Bellevue 98005
Tel: (206) 453-5874
TWX: 910-443-2469

WISCONSIN

†Arrow Electronics, Inc.
430 W. Rausson Avenue
Oakcreek 53154
Tel: (414) 764-6600
TWX: 910-262-1193

WISCONSIN (Cont'd)

Hamilton/Avnet Electronics
2975 Moorland Road
New Berlin 53151
Tel: (414) 784-4510
TWX: 910-262-1182

CANADA

ALBERTA

Hamilton/Avnet Electronics
2816 21st Street N.E.
Calgary T2E 6Z2
Tel: (403) 230-3586
TWX: 03-827-642

Hamilton/Avnet Electronics
6845 Rexwood Road Unit 6
Mississauga, Ontario L4V1R2
Tel: (416) 677-0484

Zentronics
Bay No. 1
3300 14th Avenue N.E.
Calgary T2A 6J4
Tel: (403) 272-1021

BRITISH COLUMBIA

Hamilton/Avnet Electronics
105-2550 Boundry Road
Burmalay V5M 3Z3
Tel: (604) 272-4242

Zentronics
108-11400 Bridgeport Road
Richmond V6X 1T2
Tel: (604) 273-5575
TWX: 04-5077-89

MANITOBA

Zentronics
590 Berry Street
Winnipeg R3H 0S1
Tel: (204) 775-8661

ONTARIO

Arrow Electronics Inc.
24 Martin Ross Avenue
Downsview M3J 2K9
Tel: (416) 661-0220
TELEX: 06-218213

Arrow Electronics Inc.
148 Colonnade Road
Nepean K2E 7J5
Tel: (613) 226-6903

†Hamilton/Avnet Electronics
6845 Rexwood Road
Units G & H
Mississauga L4V 1R2
Tel: (416) 677-7432
TWX: 610-492-8867

†Hamilton/Avnet Electronics
210 Colonnade Road South
Nepean K2E 7L5
Tel: (613) 226-1700
TWX: 05-349-71

†Zentronics
8 Tilbury Court
Brampton L6T 3T4
Tel: (416) 451-9600
TWX: 06-976-78

Zentronics
564/10 Weber Street North
Waterloo N2L 5C6
Tel: (519) 884-5700

Zentronics
155 Colonnade Road
Unit 17
Nepean K2E 7K1
Tel: (613) 225-8840
TWX: 06-976-78

QUEBEC

Arrow Electronics Inc.
4050 Jean Talon Quest
Montreal H4P 1W1
Tel: (514) 735-5511
TELEX: 05-25596

Arrow Electronics Inc.
909 Charest Blvd.
Quebec 61N 269
Tel: (418) 687-4231
TLX: 05-13388

Hamilton/Avnet Electronics
2795 Rue Halpern
St. Laurent H4S 1P8
Tel: (514) 335-1000
TWX: 610-421-3731

Zentronics
505 Locke Street
St. Laurent H4T 1X7
Tel: (514) 735-5361
TWX: 05-827-535

†Microcomputer System Technical Demonstrator Centers

CG-11/6/86

EUROPEAN SALES OFFICES

BELGIUM

Intel Corporation S.A.
Parc Seny
Rue du Moulin a Papier 51
Boite 1
B-1160 Brussels
Tel: (02) 661 07 11
TELEX: 24814

DENMARK

Intel Denmark A/S*
Glentevej 61 - 3rd Floor
DK-2400 Copenhagen
Tel: (01) 19 80 33
TELEX: 19567

FINLAND

Intel Finland OY
Ruosilantie 2
000390 Helsinki
Tel: (0) 544 644
TELEX: 123 332

FRANCE

Intel Paris
1 Rue Edison, BP 303
78054 Saint-Quentin en Yvelines
Tel: (33) 1 30 57 70 00
TELEX: 69901677

FRANCE (Cont'd)

Intel Corporation, S.A.R.L.
Immeuble BBC
4 Quai des Etroits
69005 Lyon
Tel: (7) 842 40 89
TELEX: 305153

WEST GERMANY

Intel Semiconductor GmbH*
Seidlstrasse 27
D-8000 Munchen 2
Tel: (89) 53891
TELEX: 05-23177 INTL D

Intel Semiconductor GmbH*
Mainzerstrasse 75
D-6200 Wiesbaden 1
Tel: (6121) 70 08 74
TELEX: 04168183 INTW D

Intel Semiconductor GmbH
Bruckstrasse 61
7012 Fellbach
Stuttgart
Tel: (711) 58 00 82
TELEX: 7254826 INTS D

Intel Semiconductor GmbH*
Hohenzollernstrasse 5*
3000 Hannover 1
Tel: (511) 34 40 81
TELEX: 923625 INTH D

ISRAEL

Intel Semiconductors Ltd*
Atidim Industrial Park
Neve Sharet
Dvora Hanevia
Bldg. No. 13, 4th Floor
P.O. Box 43202
Tel Aviv 61430
Tel: 3-491099/8
TELEX: 371215

ITALY

Intel Corporation Italia Spa*
Milanofiori, Palazzo E/3
20094 Assago (Milano)
Tel: (02) 824 40 71
TELEX: 315183 INTMIL

NETHERLANDS

Intel Semiconductor Nederland B.V.*
Alexanderpoort Building
Marten Meesweg 93
3068 Rotterdam
Tel: (10) 21 23 77
TELEX: 22283

NORWAY

Intel Norway A/S
P.O. Box 92
Hvamveien 4
N-2013
Skjetten
Tel: (2) 742 420
TELEX: 18018

SPAIN

Intel Iberia
Calle Zurbaran
28-1-IZQDA
28010 Madrid
Tel: (34) 1410 40 04
TELEX: 46880

SWEDEN

Intel Sweden A.B.*
Dalvagen 24
S-171 36 Solna
Tel: 8/7340100
TELEX: 12261

SWITZERLAND
Intel Semiconductor A.G.*
Talackerstrasse 17
8152 Glattbrugg postfach
CH-8065 Zurich
Tel: (01) 829 29 77
TELEX: 57989 ICH CH

UNITED KINGDOM

Intel Corporation (U.K.) Ltd.*
Pipers Way
Swindon, Wiltshire SN3 1RJ
Tel: (793) 696 000
TELEX: 444447 INT SWN

*Field Application Location

EUROPEAN DISTRIBUTORS/REPRESENTATIVES

AUSTRIA

Bacher Elektronische Ges.m.b.H.
Meidlinger Hauptstrasse 78
A 1120 Wien
Tel: (222) 83 56 46
TELEX: 11532 BASAT A

Moor Ges.m.b.H.
Storchengasse 1/1/1
A-1150 Wien
Tel: 222-85 86 46

BELGIUM

S.A. Inelco Belgium
Ave. des Croix de Guerre 94
B1120 Brussels
Tel: (02) 216 01 60
TELEX: 25441

ITT Electronic Components
rue Colonel Bourgstr. 105a (Bte 3)
B-1140 Brussels
Tel: (02) 735-7125

DENMARK

ITT MultiKomponent A/S
Naverland 29
DK-2600 Glostrup
Tel: (02) 456 66 45
TX: 33355

FINLAND

Oy Fintronic AB
Melkonkatu 24 A
SF-00210 Helsinki 21
Tel: (0) 692 60 22
TELEX: 124 224 Ftron SF

FRANCE

Generim
Z.A. de Courtsboeuf
Avenue de la Baltique
F-91943 Les Ulis Cedex-B.P. 88
Tel: (1) 69 07 78 78
TELEX: F691700

Jermyn
16, Avenue de Jean-Jaures
94600 Choisy-Le-Roi
Tel: (1) 48 53 12 00
TELEX: 260 967

Metrologie
Tour d' Asnieres
4, Avenue Laurent Cely
92606-Asnieres
Tel: (1) 47 90 62 40
TELEX: 611-448

Tekelec Airtronic
Cite des Bruyeres
Rue Carle Vernet B.P. 2
92310 Sevre S
Tel: (1) 45 34 75 35
TELEX: 204552

WEST GERMANY

Electronic 2000 Vertriebs A.G.
Stahlgruberring 12
8000 Munich 82
Tel: (89) 42 00 10
TELEX: 522561 EEC D

Jermyn GmbH
Postfach 11 80
Schulstrasse 84
6277 Bad Camberg
Tel: (06434) 231
TELEX: 484426 JERM D

CES Computer Electronics Systems GmbH
AM Moosfeld 37
8000 Munich 82
Tel: (089) 420-430
TELEX: 2180260

Metrologie GmbH
Hanaastrasse 15
8000 Munich 21
Tel: (89) 570-940
TELEX: 5213189

Proelectron Vertriebs AG
Max Planck Strasse 1-3
6072 Dreieich
Tel: (6103) 3040
TELEX: 417983

ITT-Multikomponent
Bahnhofstrasse 44
7141 Moeglingen
Tel: (07141) 4870

IRELAND

Micro Marketing
Glenageary Office Park
Glenageary
Co. Dublin
Tel: (1) 85 62 88
TELEX: 31584

ISRAEL

Eastronics Ltd.
11 Rozanis Street
P.O. Box 39300
Tel Aviv 61392
Tel: (3) 47 51 51
TELEX: 33638

ITALY

Eledtra 3S S.P.A.
Via G. Watt, 37
I 20143 Milano
Tel: (2) 81 82 1
TELEX: 332332

ITALY (Cont'd)

Intesi
Milanofiori E/5
20090 Assago
Tel: (2) 824701
TELEX: 311351

Lasi Elettronica S.P.A.
V. Le Fulvio Testi, 126
20092 Cinisello Balsamo (Milano)
Tel: 02/2440012
TELEX: 352040 LASIMI I
TELEFAX: 2487717

NETHERLANDS

Koning & HartmanElectrotechniek B.V.
P.O. Box 125
2600 AC Delft
Tel: 15 609-906
TELEX: 31528

NORWAY

Nordisk Elektronic A/S
Post Office Box 130
N-1364 Hvalstad
Tel: (2) 846 210
TELEX: 17546

PORTUGAL

Ditram
Avenida Miguel Bombarda, 133
P-1000 Lisboa
Tel: (1) 154 53 13
TELEX: 14182 Brieks-P

SPAIN

ITT SESA
Miguel Angel 21-3
Madrid 28010
Tel: (1) 419 54 00
TELEX: 27461

Diode
Calle Gandesa 10-4
08028 Barcelona
Tel: (3) 322-12-51
TELEX: 42148

SWEDEN

Nordisk Electronik AB
Huvudstagatan 1
Box 1409
S-171 27 Solna
Tel: (8) 734 97 70
TELEX: 10547

SWITZERLAND

Industrade AG
Hertistrasse
CH-8304 Wallisellen
Tel: (01) 830 50 40
TELEX: 56788 INDEL CH

UNITED KINGDOM

Accent Electronic Components Ltd.
Jubilee House
Jubilee Way
Letchworth
Hertfordshire SG6 1QH
Tel: (0462) 686666
TELEX: 826293

Bytech Ltd.
2 The Western Center
Western Industrial Estate
Bracknell
Berkshire RG12 1RW
Tel: (0344) 482211
TELEX: 848215

Comway Microsystems Ltd.
Market Street
Bracknell
Berkshire
Tel: (344) 55333
TELEX: 847201

IBR Microcomputers Ltd.
Unit 2 Western Center
Western Industrial Estate
Bracknell
Berkshire RG12 1RW
Tel: (0344) 486555
TELEX: 849381

Jermyn Industries Vestry Estate
Oxford Road
Seven Oaks
Kent TN 14 5EU
Tel: (0732) 450144
TELEX: 95142

M.E.D.L.
East Lane Road
North Wembley
Middlesex HA9 7PP
Tel: (1)904 49307
TELEX: 28817

Rapid Recall, Ltd.
Rapid House/Denmark St.
High Wycombe
Bucks HP11 2ER
Tel: (494) 26 271
TELEX: 837931

IBR
2 The Western Center
Western Road
Bracknell, Berkshire
Tel: (0344) 486-555

CG-11/6/86

UNITED STATES
Intel Corporation
3065 Bowers Avenue
Santa Clara, CA 95051

JAPAN
Intel Japan K.K.
5–6 Tokodai Toyosato-machi
Tsukuba-gun, Ibaraki-ken 300-26
Japan

FRANCE
Intel Paris
1 Rue Edison, BP 303
78054 Saint-Quentin en Yvelines
France

UNITED KINGDOM
Intel Corporation (U.K.) Ltd.
Piper's Way
Swindon
Wiltshire, England SN3 1RJ

WEST GERMANY
Intel Semiconductor GmbH
Seidlstrasse 27
D-8000 Munchen 2
West Germany

HONG KONG
Intel Semiconductor Ltd.
1701 Connaught Centre
1 Connaught Road
Hong Kong

Printed in U.S.A. 0187 78K RRD SM
Embedded Controller

ISBN 1-55512-060-1